CS4102 Algorithms

Spring 2020

Today's Keywords

- Reductions
- Bipartite Matching
- Vertex Cover
- Independent Set
- Decision problems, verification problems
- NP, NP-Hard, NP-Compete

CLRS Readings

- Chapter 34

Reductions

- Algorithm technique of supreme ultimate power
- Convert instance of problem A to an instance of Problem B
- Convert solution of problem B back to a solution of problem A

- Why? (You might be asking. ☺)
  - As you've seen, might be a useful way to develop solution to A
  - Also, lower-bounds proofs
    - We can't find polynomial solutions to some problems.
      We want to know if they are really exponential!

MacGyver's Reduction

Problem we don't know how to solve

A

Opening a door

Aim duct at door, insert keg

Problem we do know how to solve

B

Lighting a fire

How?

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Put fire under the Keg

Reduction

Solution for $\boldsymbol{B}$

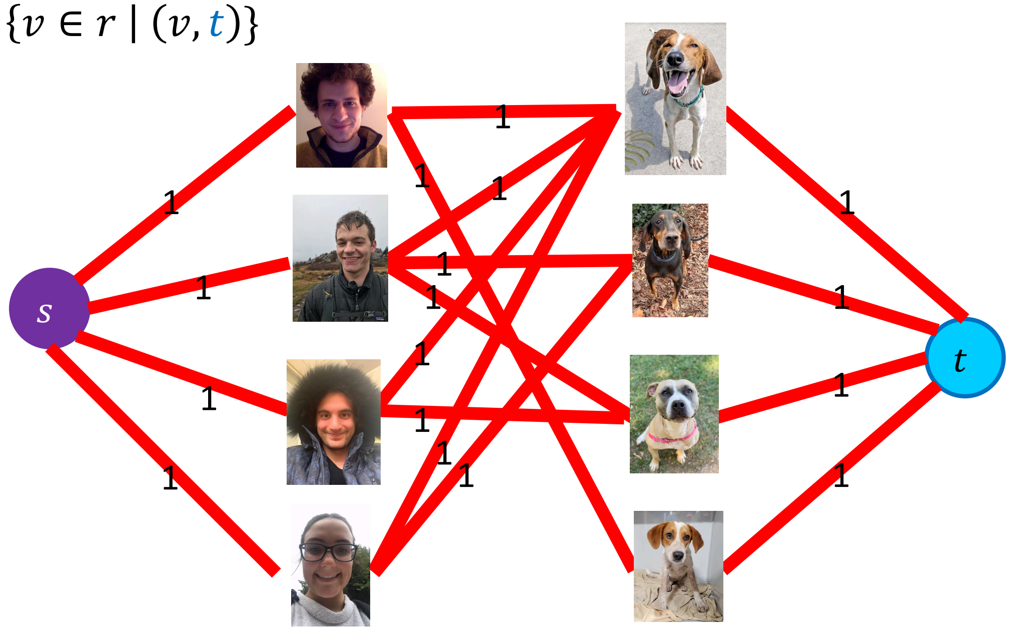
Alcohol, wood, matches

Maximum Bipartite Matching

Maximum Bipartite Matching Using Max Flow

Make $G = (L, R, E)$ a flow network $G' = (V', E')$ by:

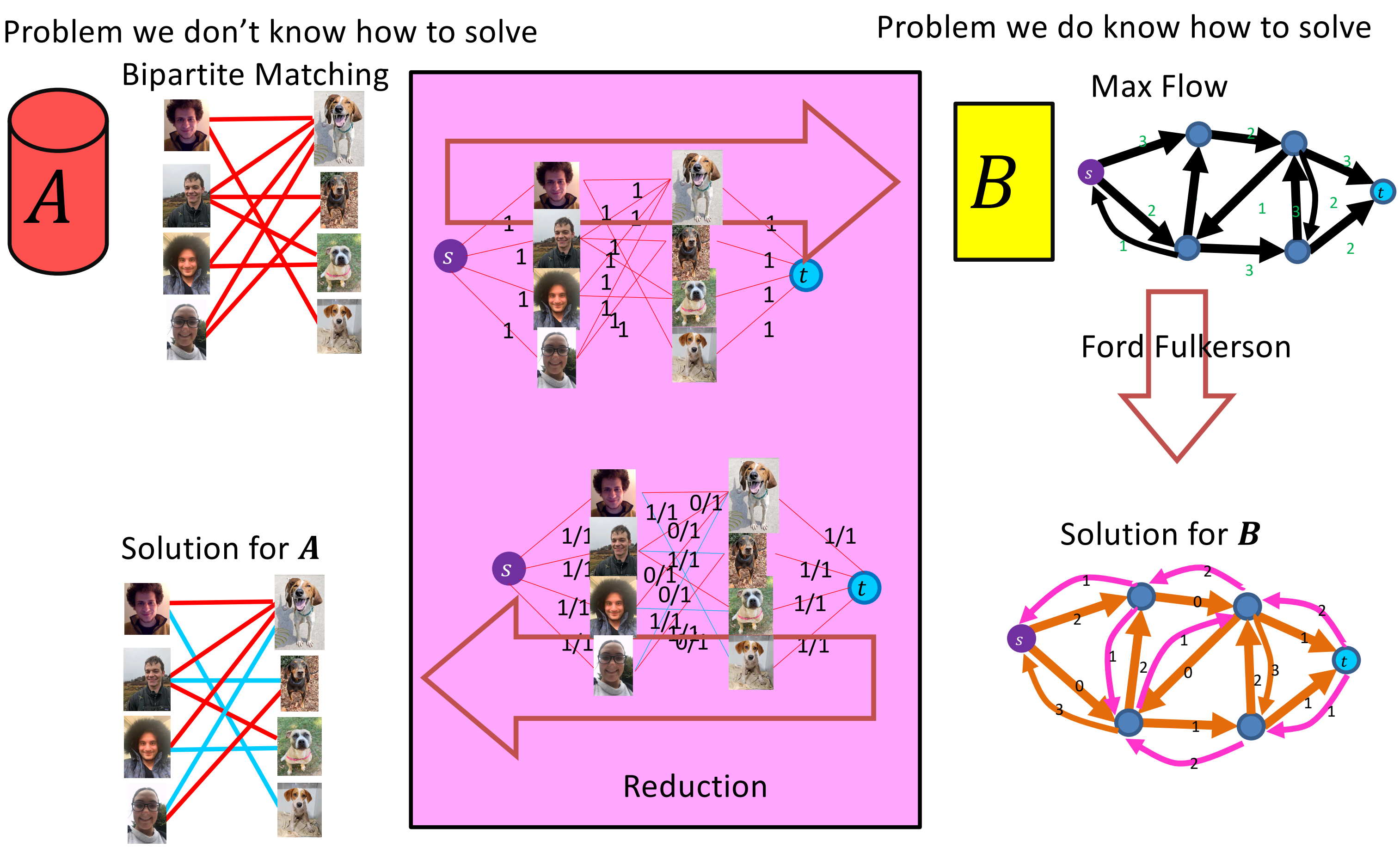
- Adding in a source and sink to the set of nodes:
  - $V' = L \cup R \cup \{s, t\}$
- Adding an edge from source to L and from R to sink:
  - $E' = E \cup \{u \in L \mid (s, u)\} \cup \{v \in r \mid (v, t)\}$
- Make each edge capacity 1:
  - $\forall e \in E', c(e) = 1$



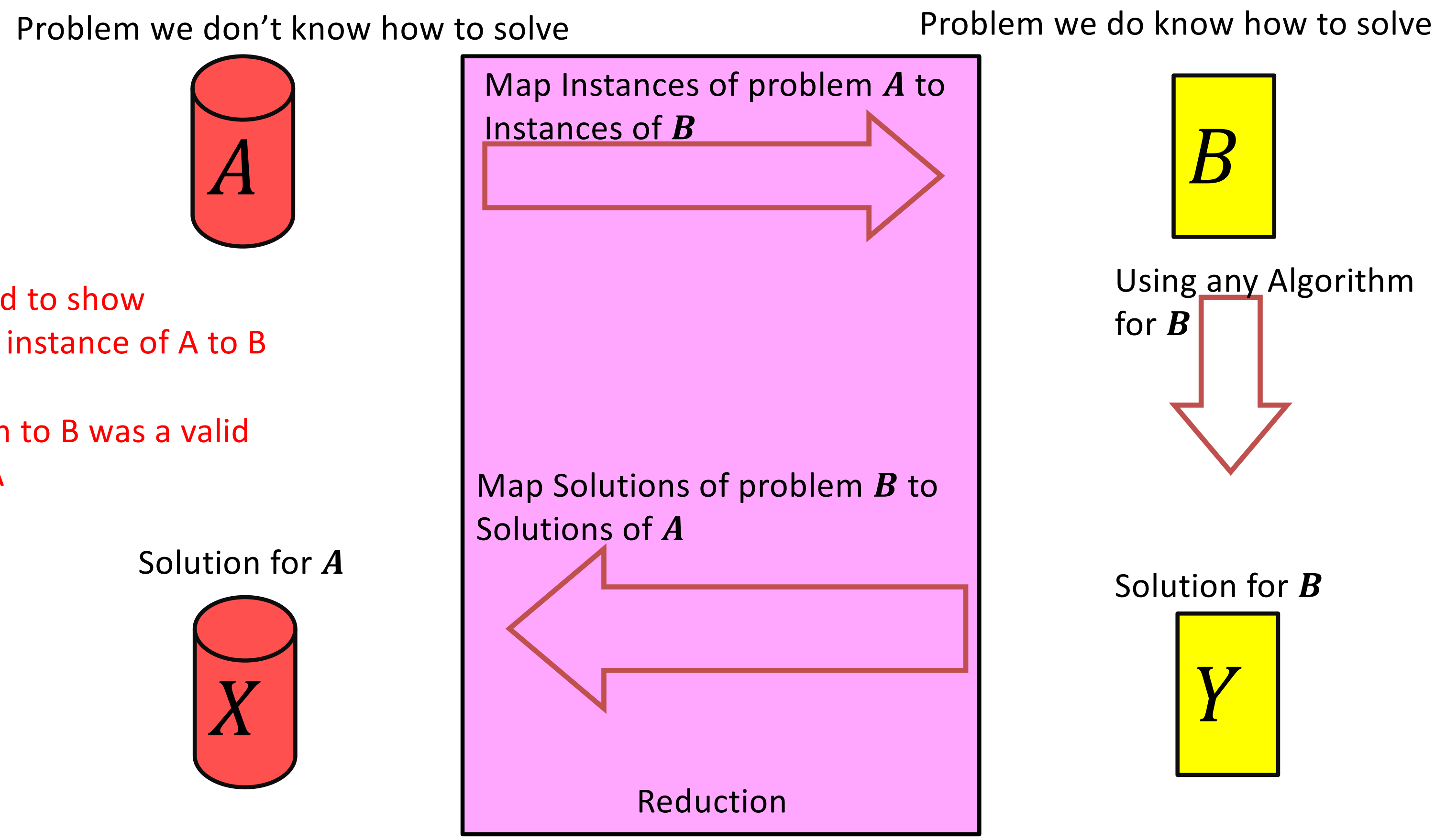
Remember: need to show

1. How to map instance of MBM to MF (and back) - construction
2. A valid solution to MF instance is a valid solution to MBM instance

Bipartite Matching Reduction



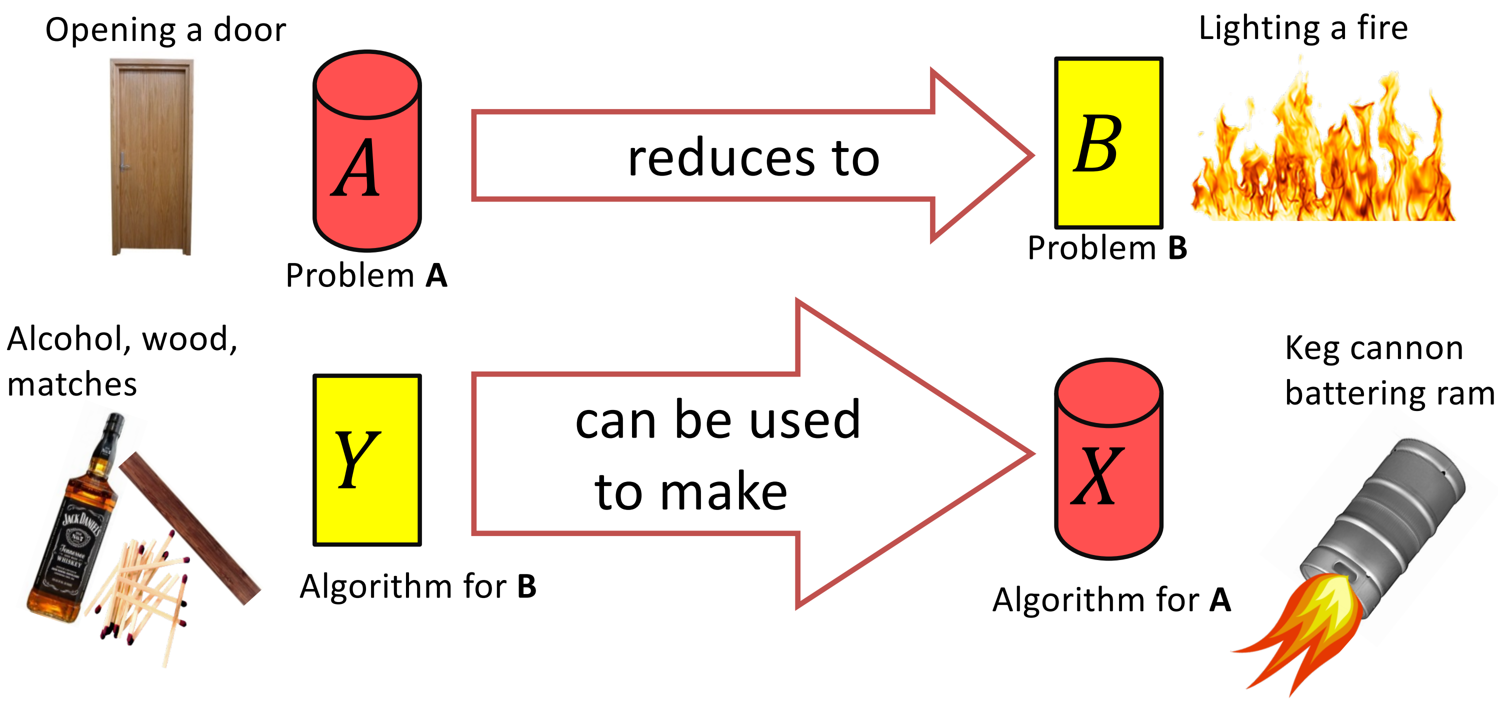
In General: Reduction



Remember: need to show

1. How to map instance of A to B (and back)
2. Why solution to B was a valid solution to A

Worst-case lower-bound Proofs

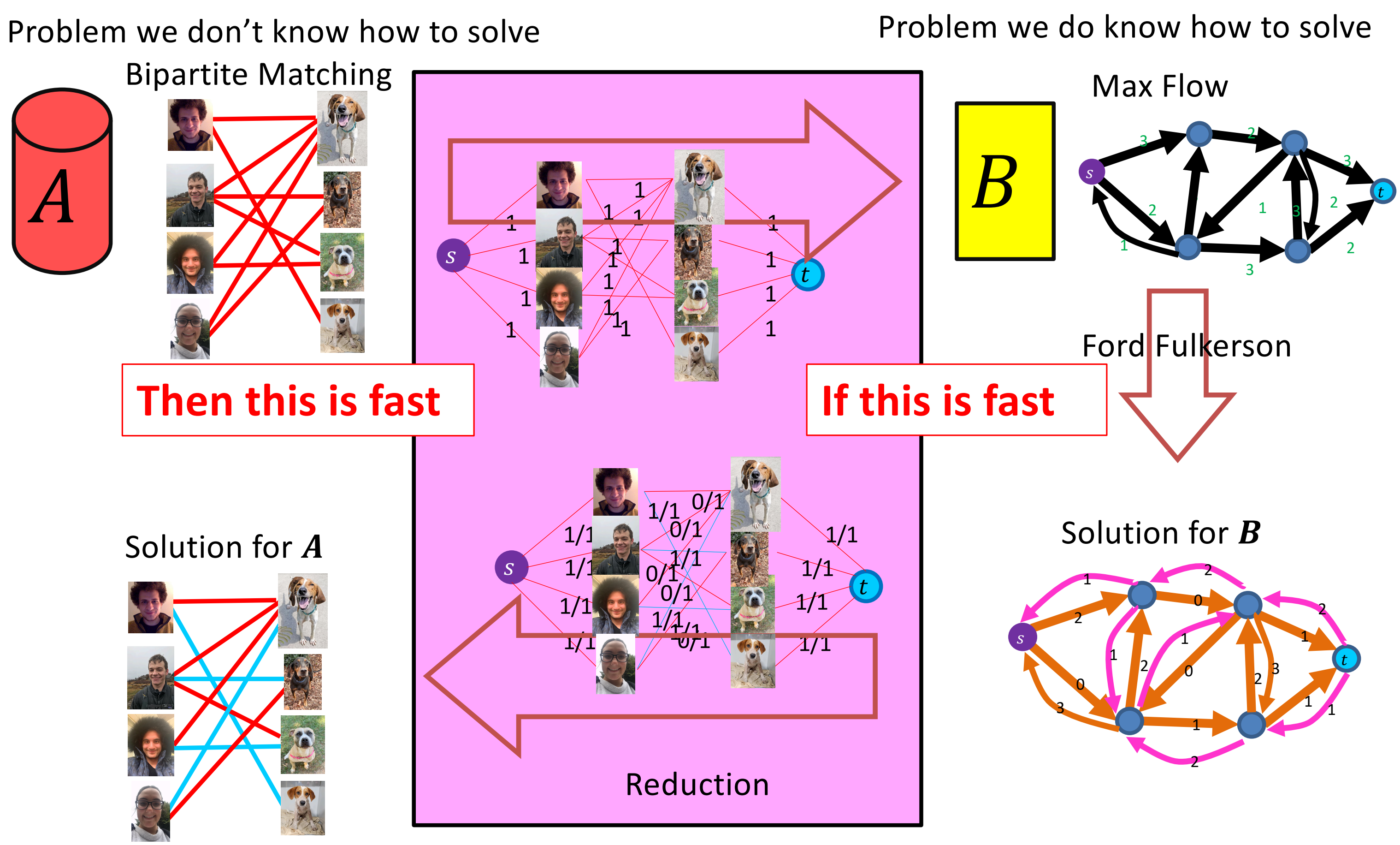


A is not a harder problem than B

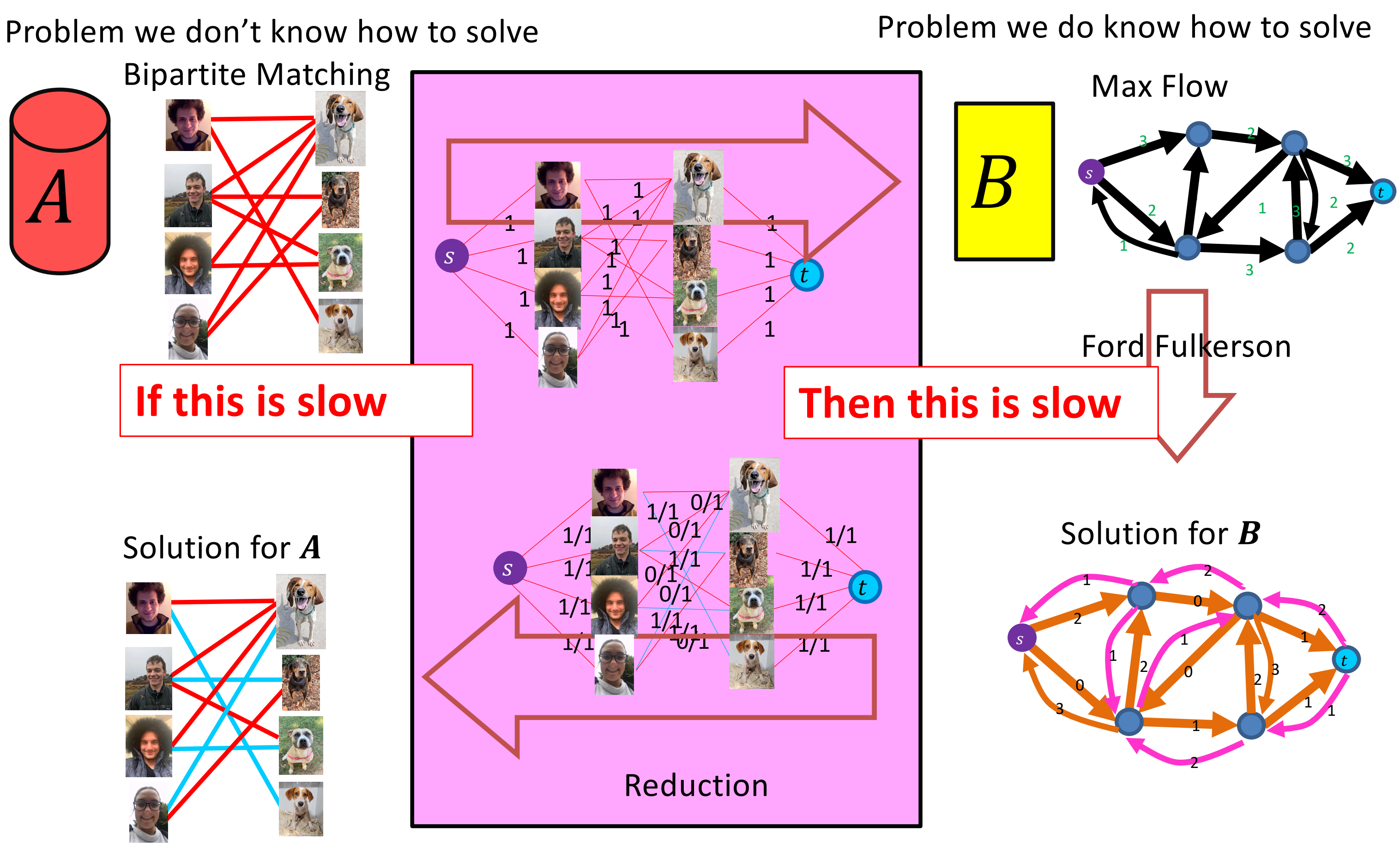
$$A \leq B$$

The name “reduces” is confusing: it is in the *opposite* direction of the making

Bipartite Matching Reduction

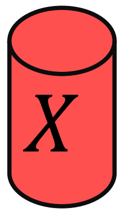


Bipartite Matching Reduction

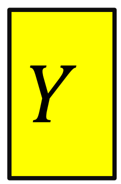


Proof of Lower Bound by Reduction

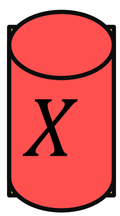
<u>To Show: Y is slow</u>

1. We know X is slow (by a proof)
(e.g., X = some way to open the door)

2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)

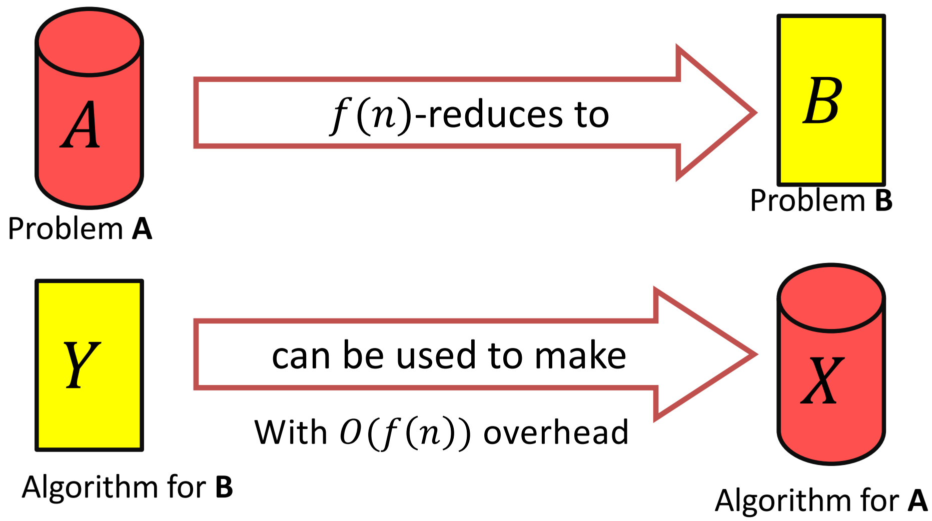
3. Show how to use Y to perform X quickly

4. X is slow, but Y could be used to perform X quickly
conclusion: Y must not actually be quick

Same Again, Different Explanation

- Say we know these two things about problems A and B:
  - First, $\boldsymbol{A} \leq \boldsymbol{B}$
  - Second, we've proven solving A is "slow" (using some lower-bounds proof)
- What can we say about B?
  - Solving B must be "slow". Why?
- Argument:
  - Assume solving B could be "fast"
  - We can solve A using B
  - That's a fast solution for A
  - But one of our givens: it's been proved A has no fast solutions. Contradiction!
  - Therefore assumption that B is "fast" is wrong. Solving B must be "slow".
    - Remember we said: A is no harder than B
- **Big point:** We can use known "slow" problems to show other problems are "slow"

Reduction Proof Notation



A is not a harder problem than B

$$A \le B$$

If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

$$A \le_{f(n)} B$$

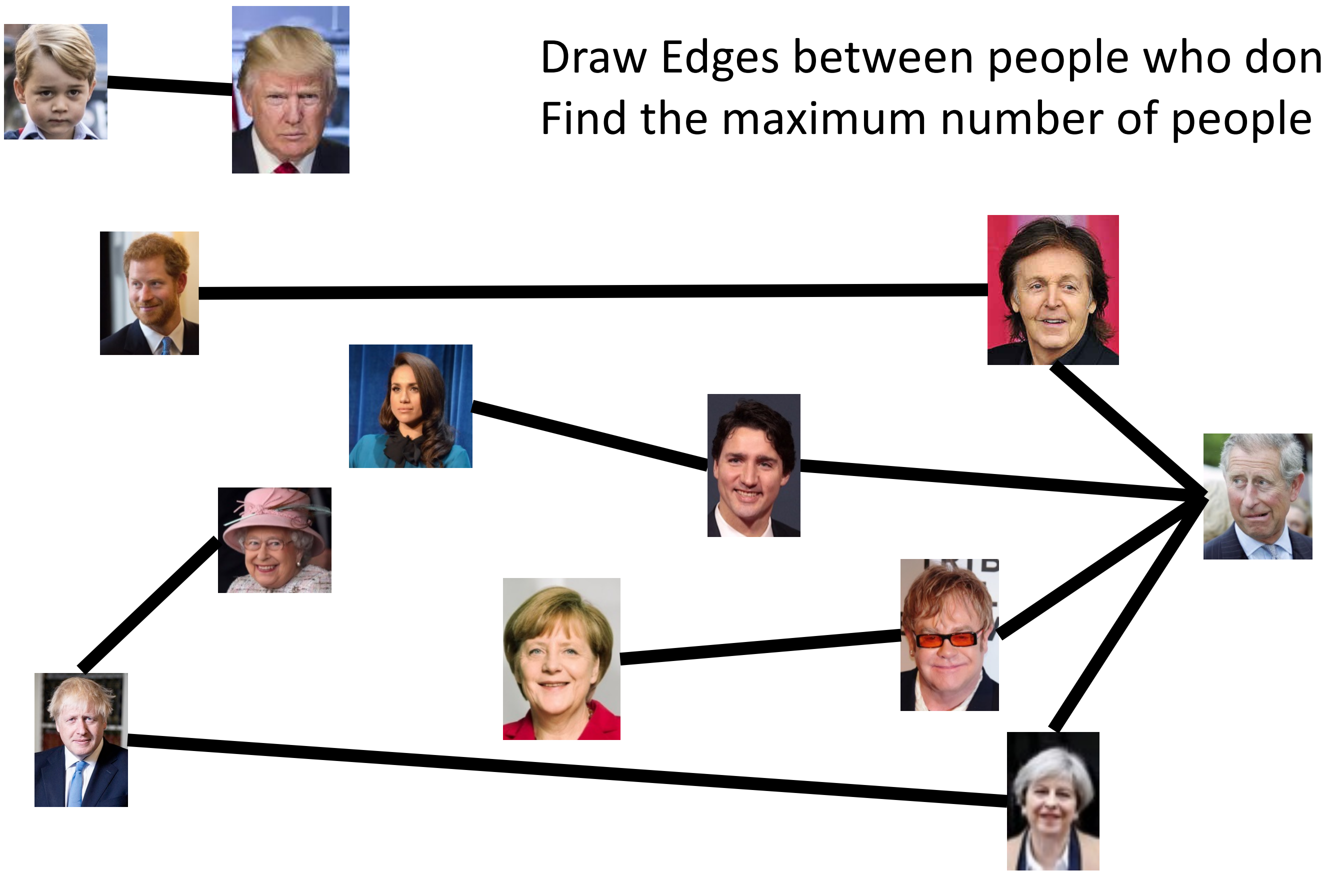
Or we could have solved A faster using B's solver!

Peek Ahead to Where We're Going

- We're going to start looking at a set of *intractable* problems
  - No known polynomial solutions have been found
  - But none have proven to require exponential time either!
- We've found polynomial reductions between a group of these (called NP-C), and we'll see that
  - None of them are "harder" than any of the others.
  - If one has a polynomial solution, they all do.
  - If there's an exponential lower-bound proof for one, all are exponential.
  - And there's more to say about these ideas later!
- Important note about discussions that follow:
  - Not showing how to solve any of these problems directly.
  - Only showing how to reduce on problem to another!

Party Problem

Draw Edges between people who don't get along
Find the maximum number of people who get along

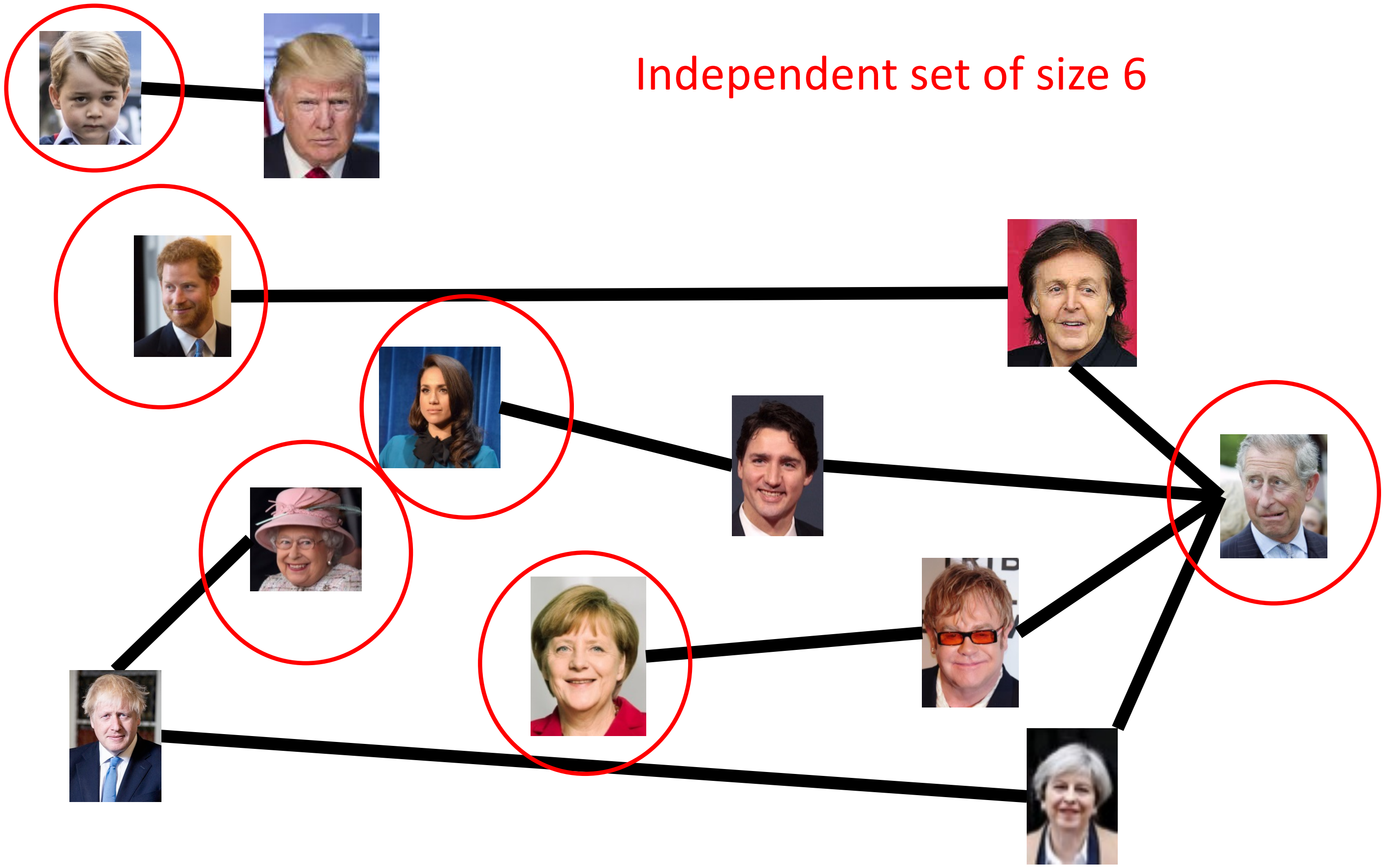


Maximum Independent Set

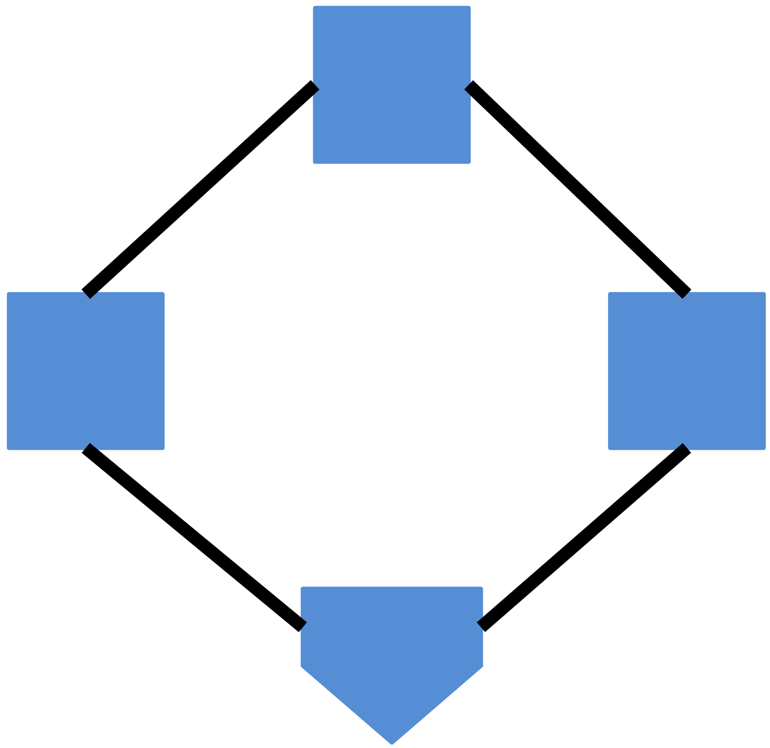
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

Example

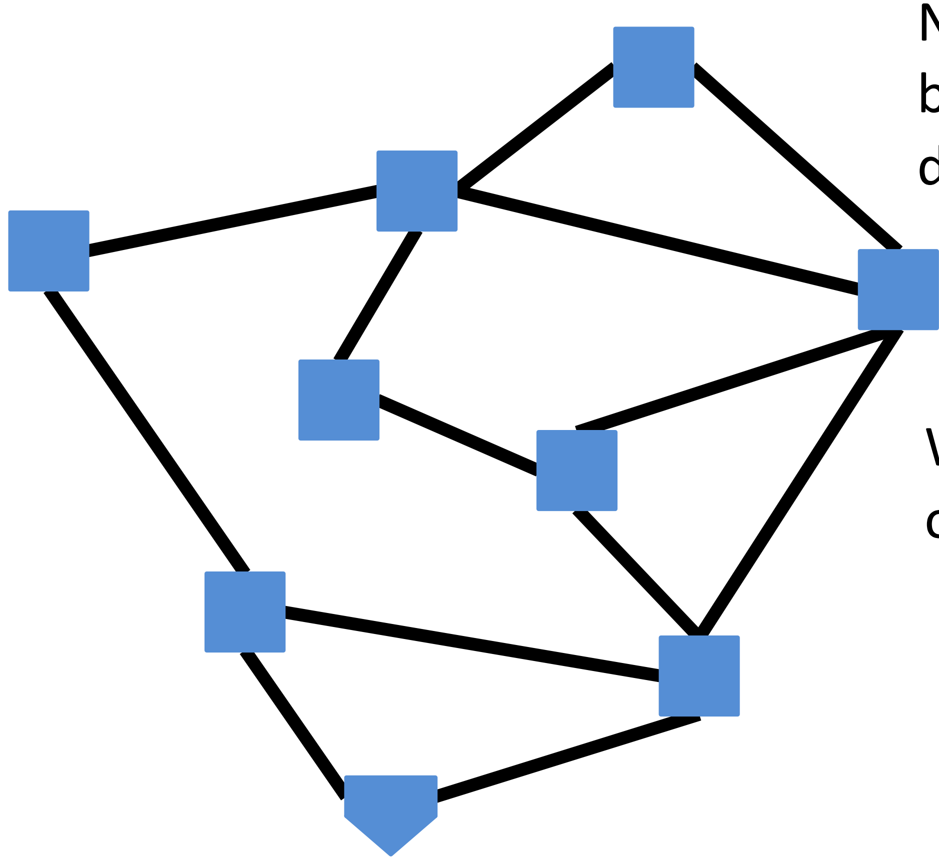
Independent set of size 6



Generalized Baseball



Generalized Baseball



Need to place defenders on bases such that every edge is defended

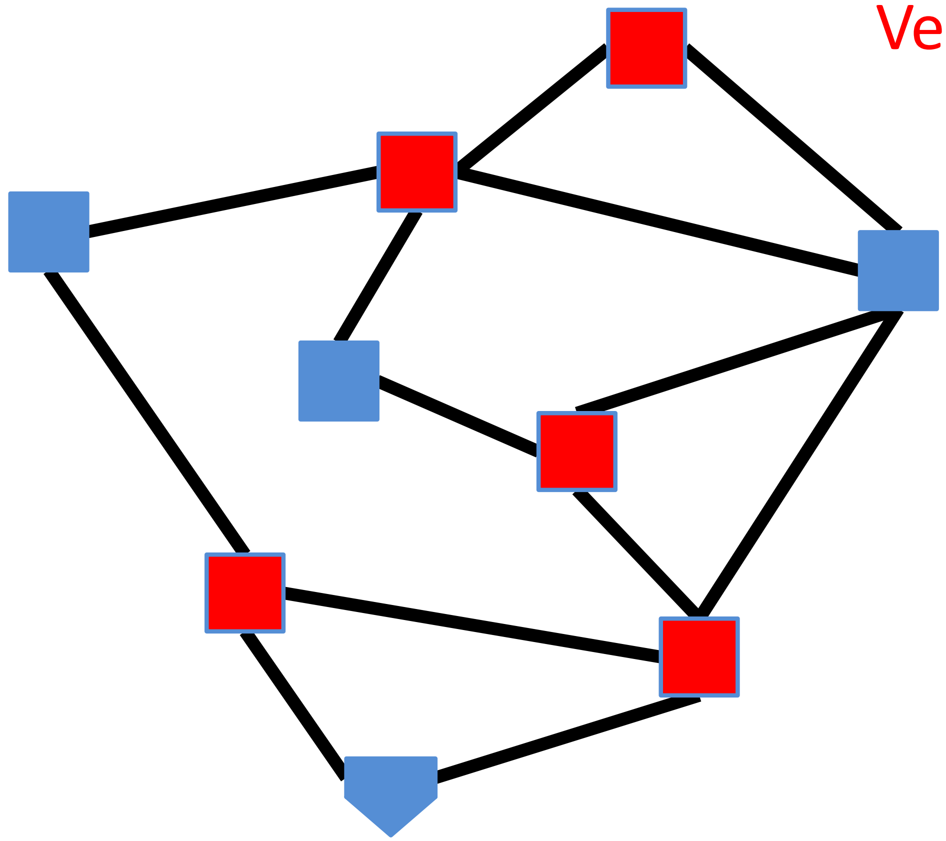
What's the fewest number of defenders needed?

Minimum Vertex Cover

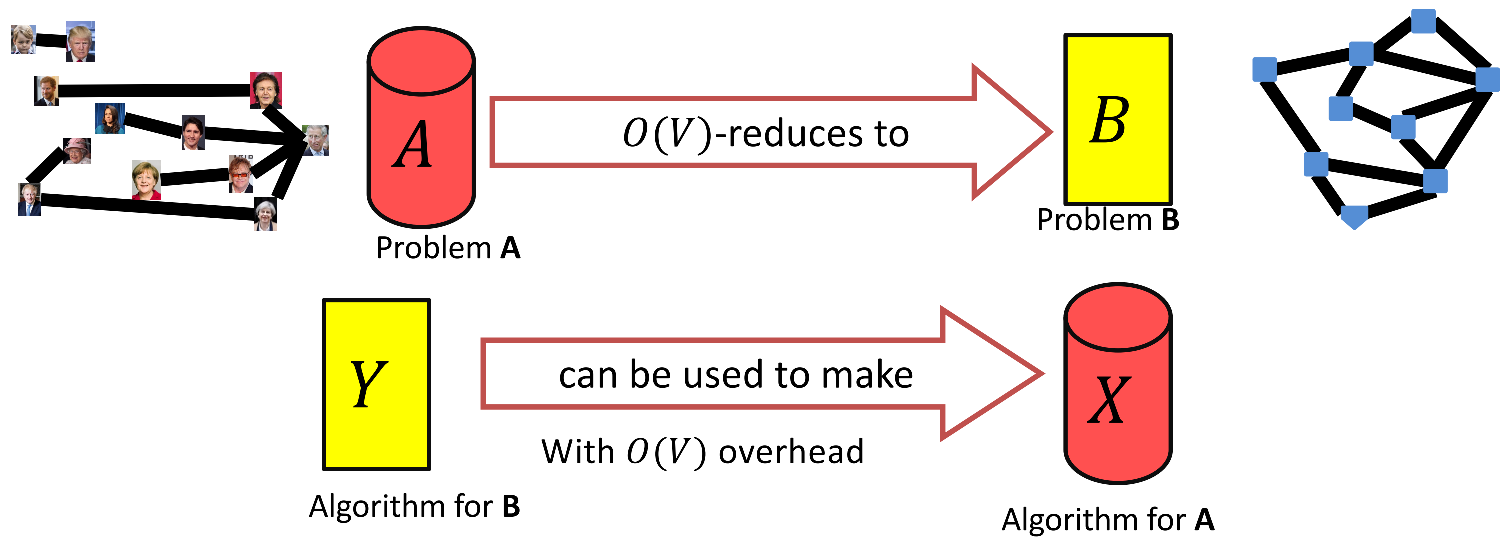
- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

Example

Vertex cover of size 5



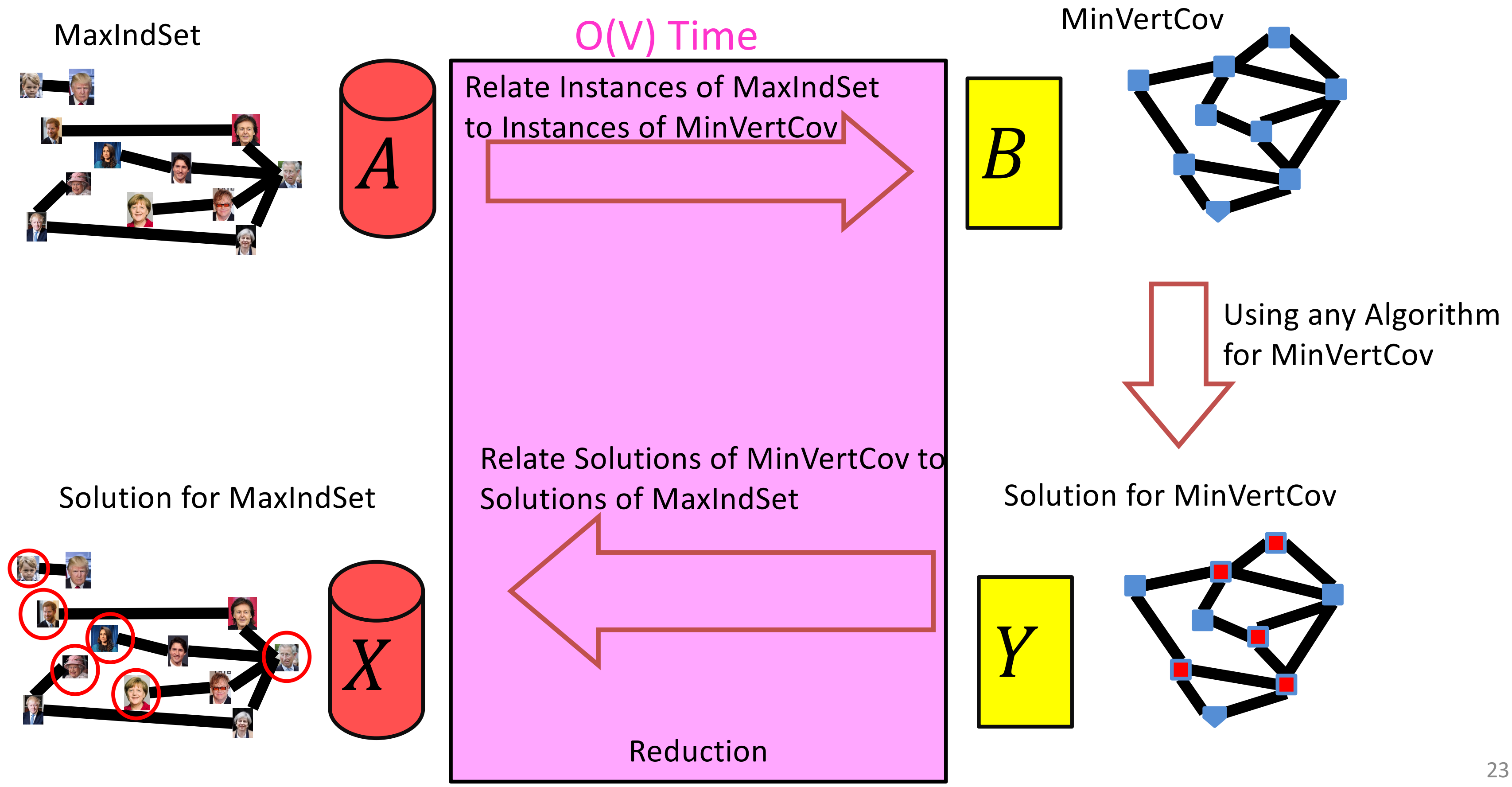
MaxIndSet$\leq_V$MinVertCov



If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

$$A \leq_V B$$

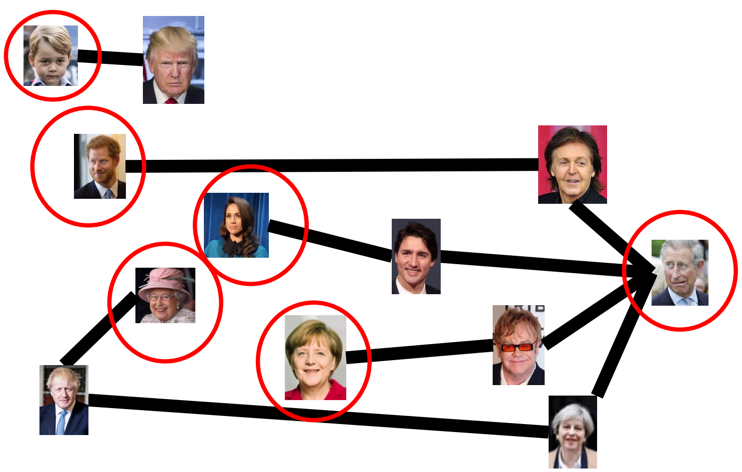
We need to build this Reduction



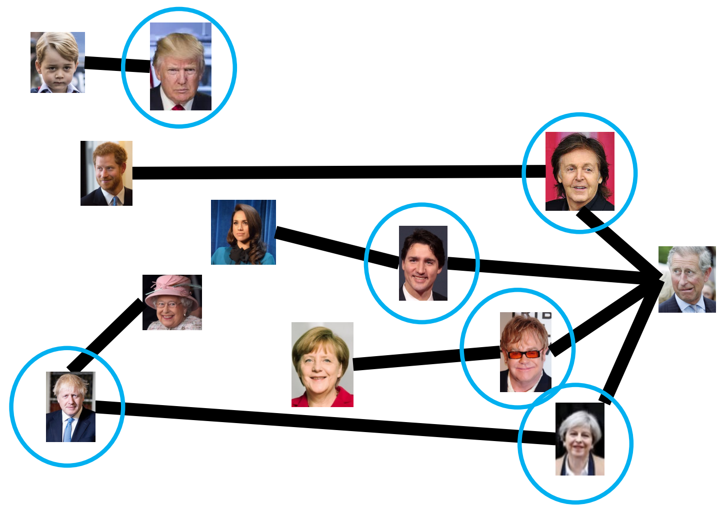
Reduction Idea

S is an independent set of G iff $V - S$ is a vertex cover of G

Independent Set



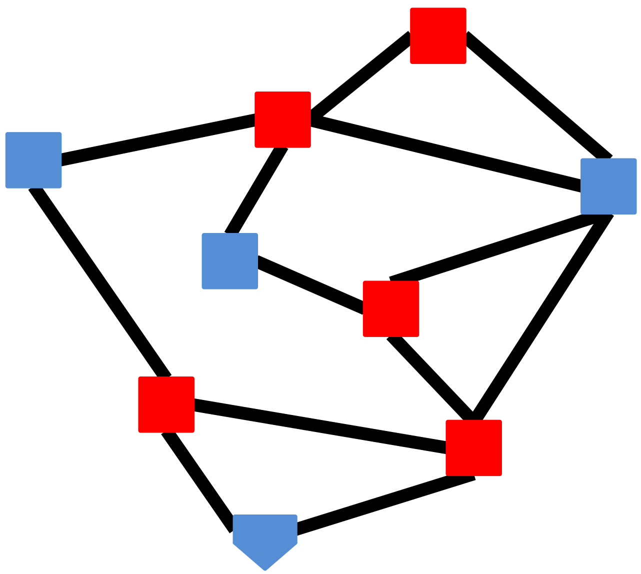
Vertex Cover



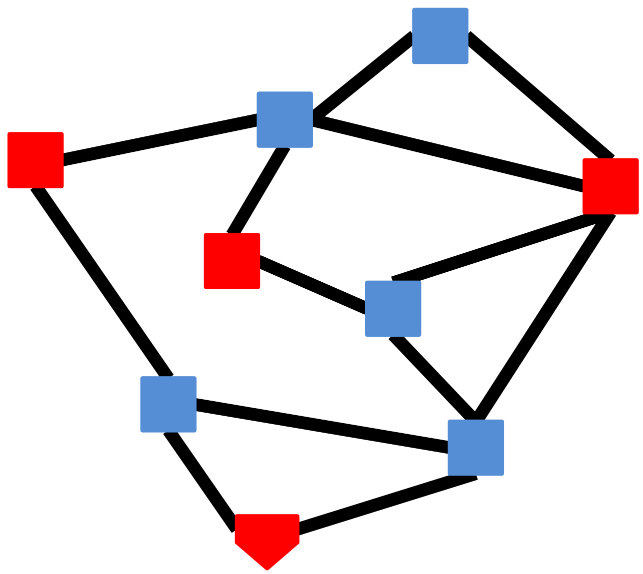
Reduction Idea

S is an independent set of G iff $V - S$ is a vertex cover of G

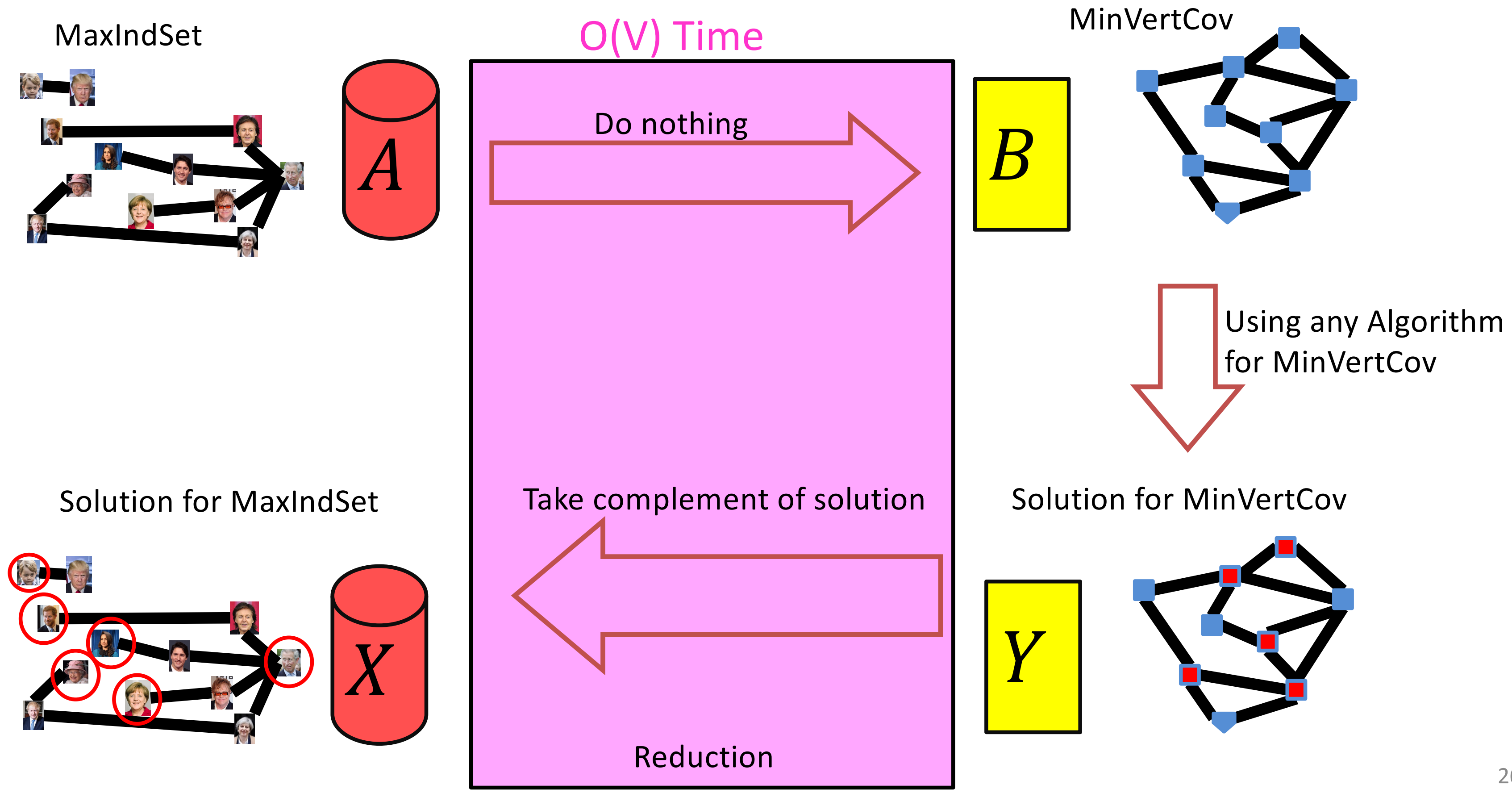
Vertex Cover



Independent Set



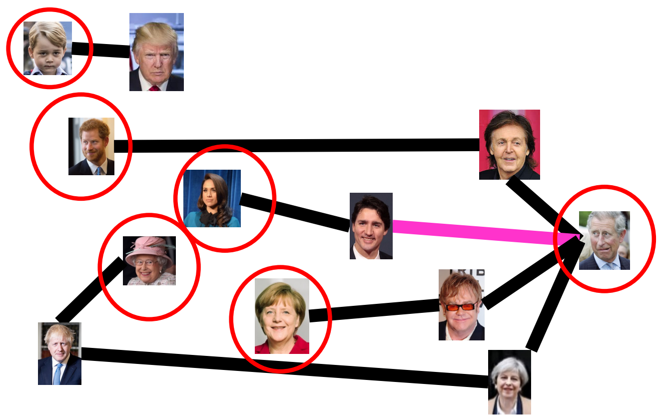
MaxIndSet V-Time Reducible to MinVertCov



Proof: $\Rightarrow$

S is an independent set of G iff $V - S$ is a vertex cover of G

Let S be an independent set



Consider any edge $(x, y) \in E$

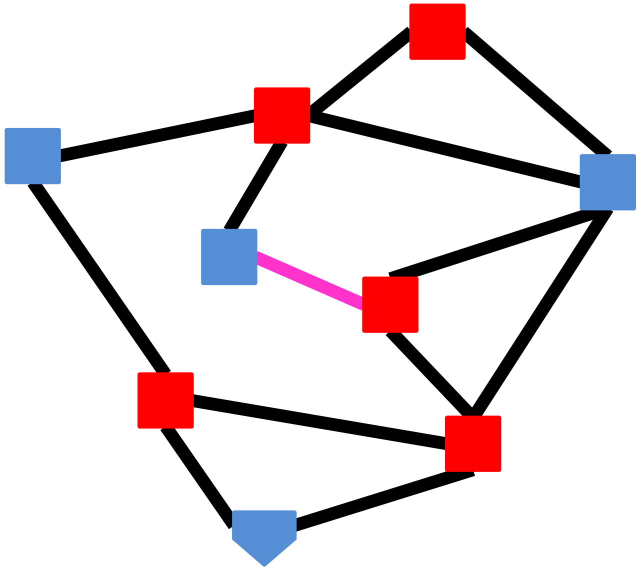
If $x \in S$ then $y \notin S$, because otherwise S would not be an independent set

Therefore $y \in V - S$, so edge (x, y) is covered by $V - S$

Proof: $\Leftarrow$

S is an independent set of G iff $V - S$ is a vertex cover of G

Let $V - S$ be a vertex cover



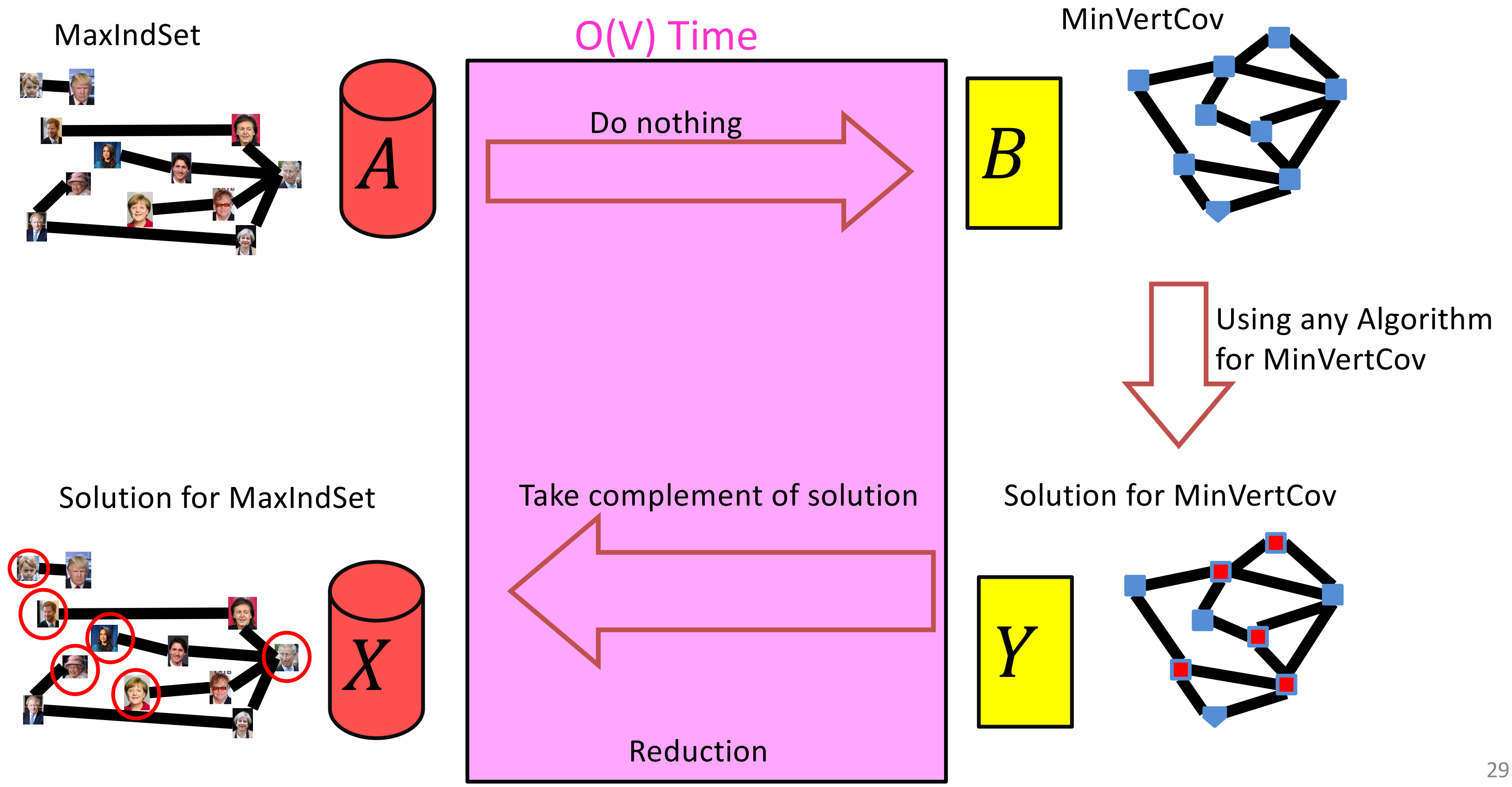
Consider any edge $(x, y) \in E$

At least one of x and y belong to $V - S$, because $V - S$ is a vertex cover

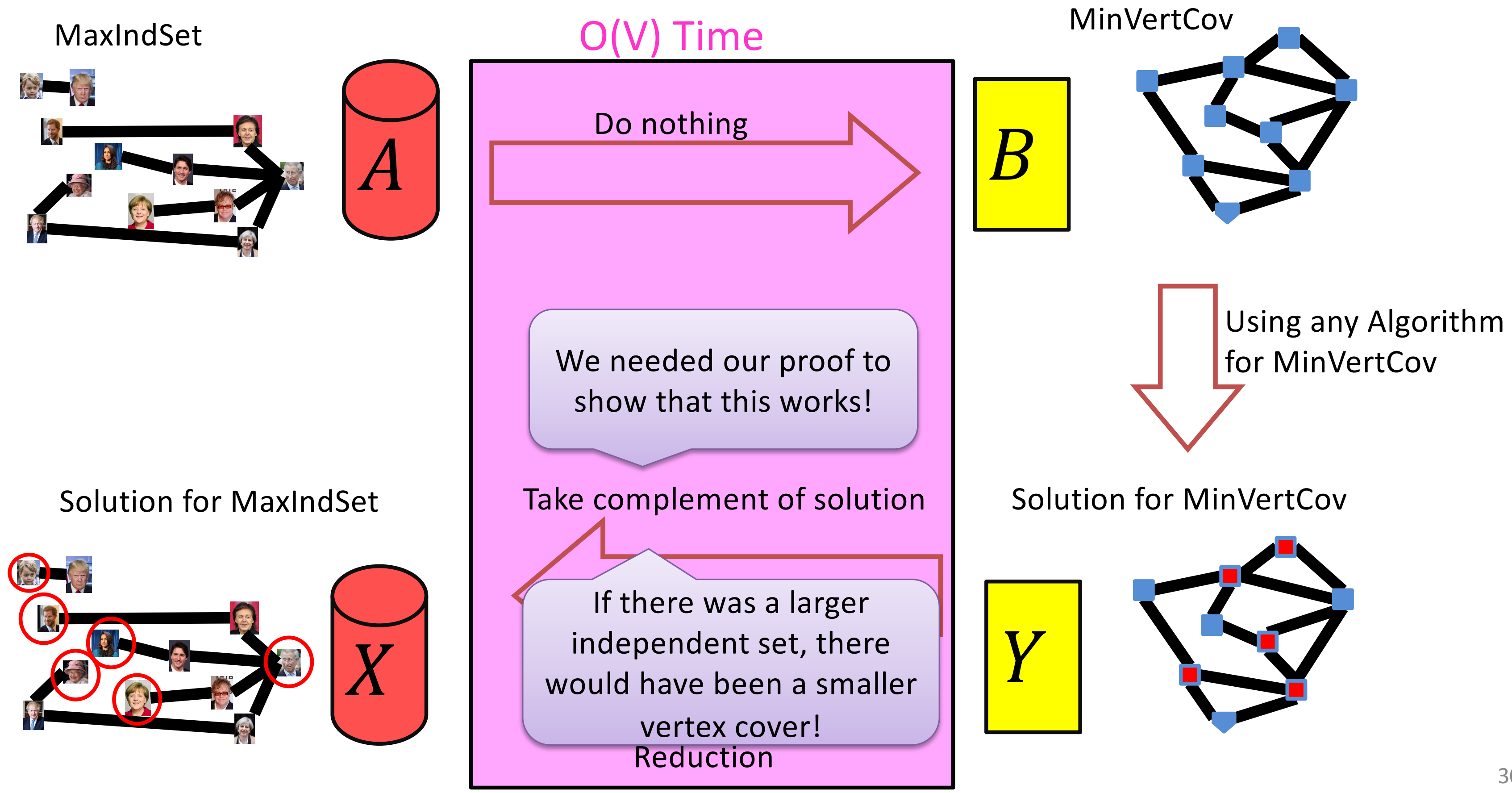
Therefore x and y are not both in S,

No edge has both end-nodes in S, thus S is an independent set

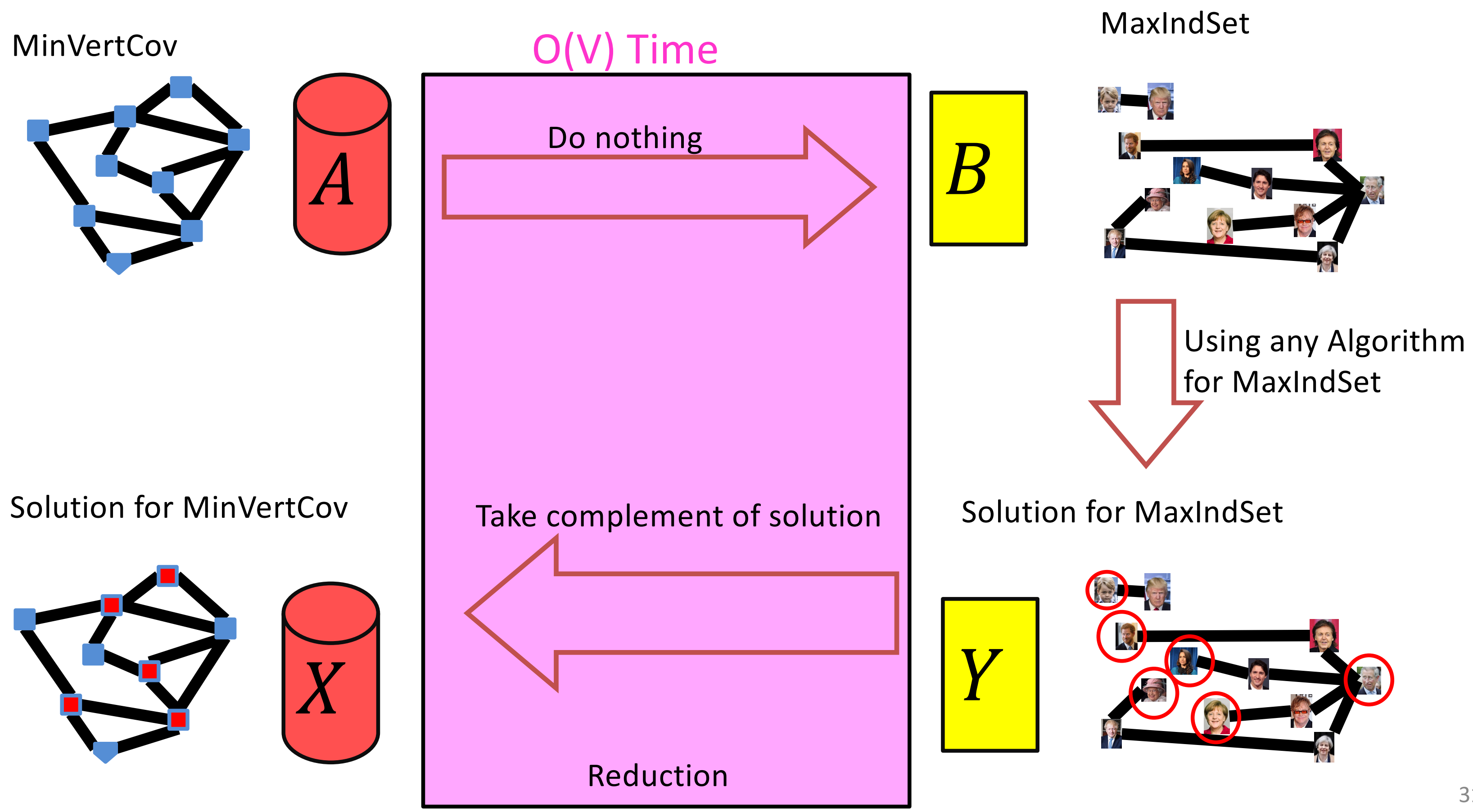
MaxIndSet V-Time Reducible to MinVertCov



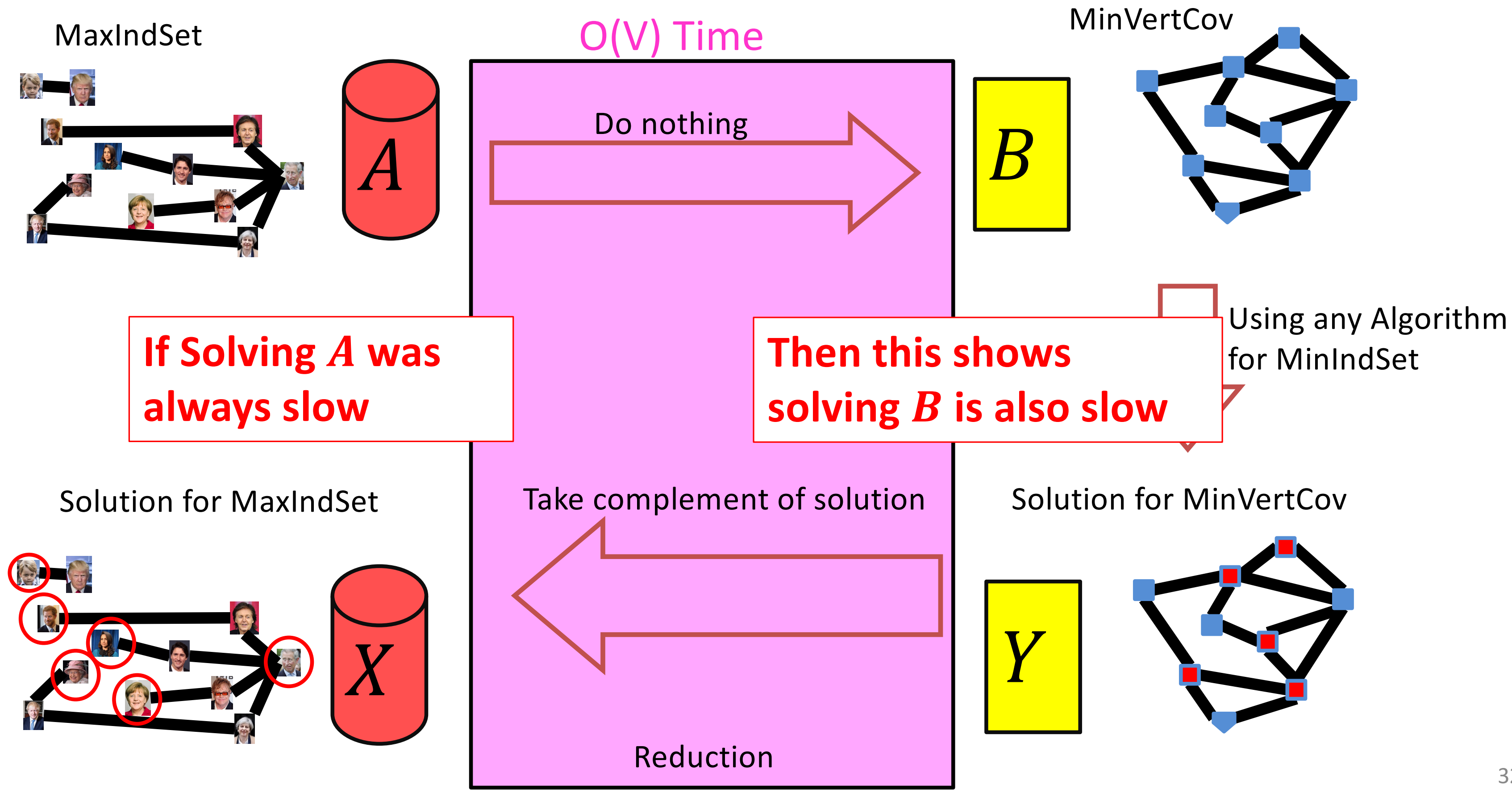
MaxIndSet V-Time Reducible to MinVertCov



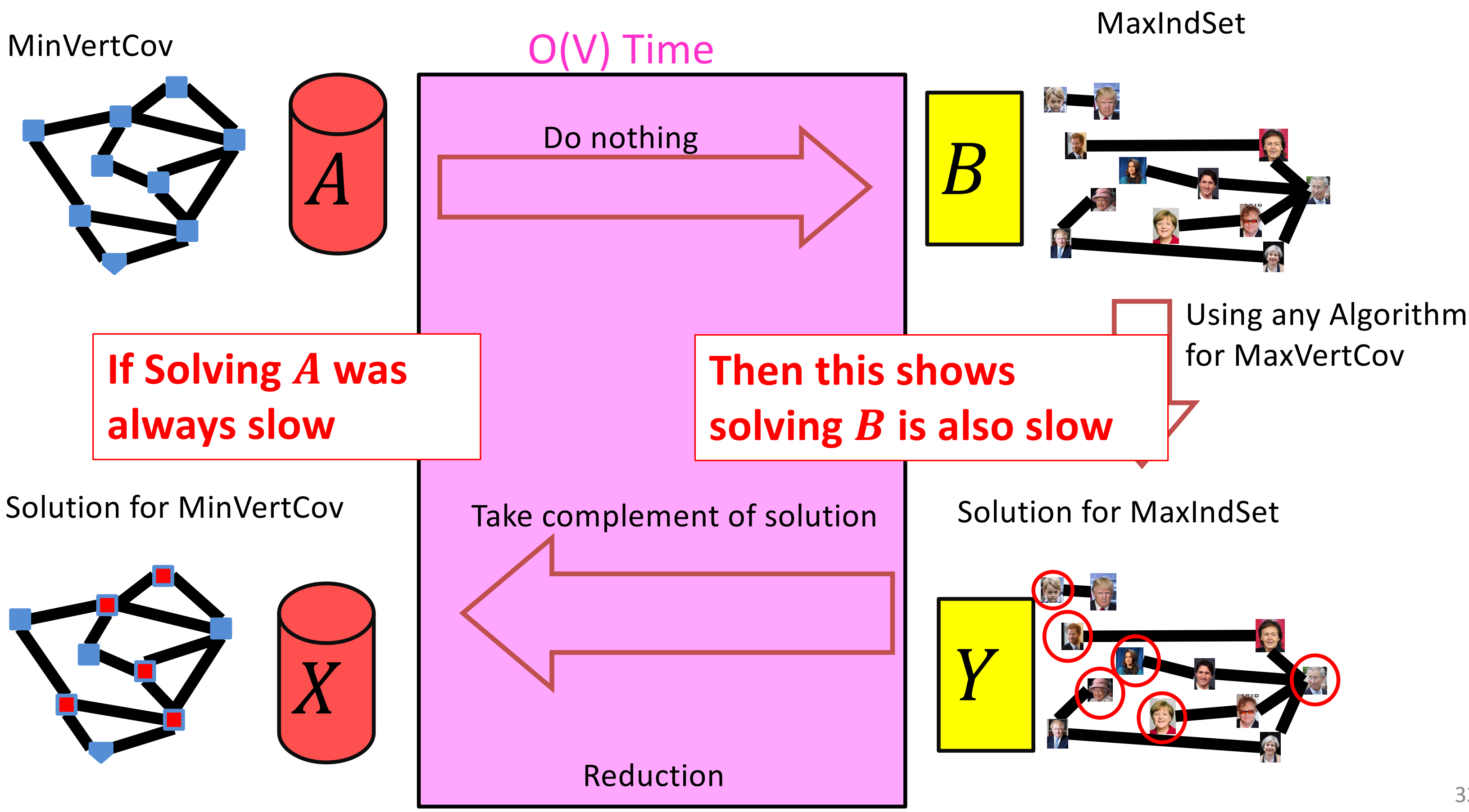
MinVertCov V-Time Reducible to MinIndSet



Corollary


MaxIndSet
A
O(V) Time
Do nothing
B
MinVertCov
Using any Algorithm
for MinIndSet
If Solving A was always slow
Then this shows solving B is also slow
Solution for MaxIndSet
Take complement of solution
Solution for MinVertCov
X
Y
Reduction

Corollary



MinVertCov
A
O(V) Time
Do nothing
B
MaxIndSet
Using any Algorithm for MaxVertCov
If Solving A was always slow
Then this shows solving B is also slow
Solution for MinVertCov
Take complement of solution
Solution for MaxIndSet
X
Y
Reduction

Conclusion

- MaxIndSet and MinVertCov are either both fast, or both slow
  - Spoiler alert: We don't know which!
    - (But we think they're both slow)
  - Both problems are NP-Complete

Mid-class warm up:

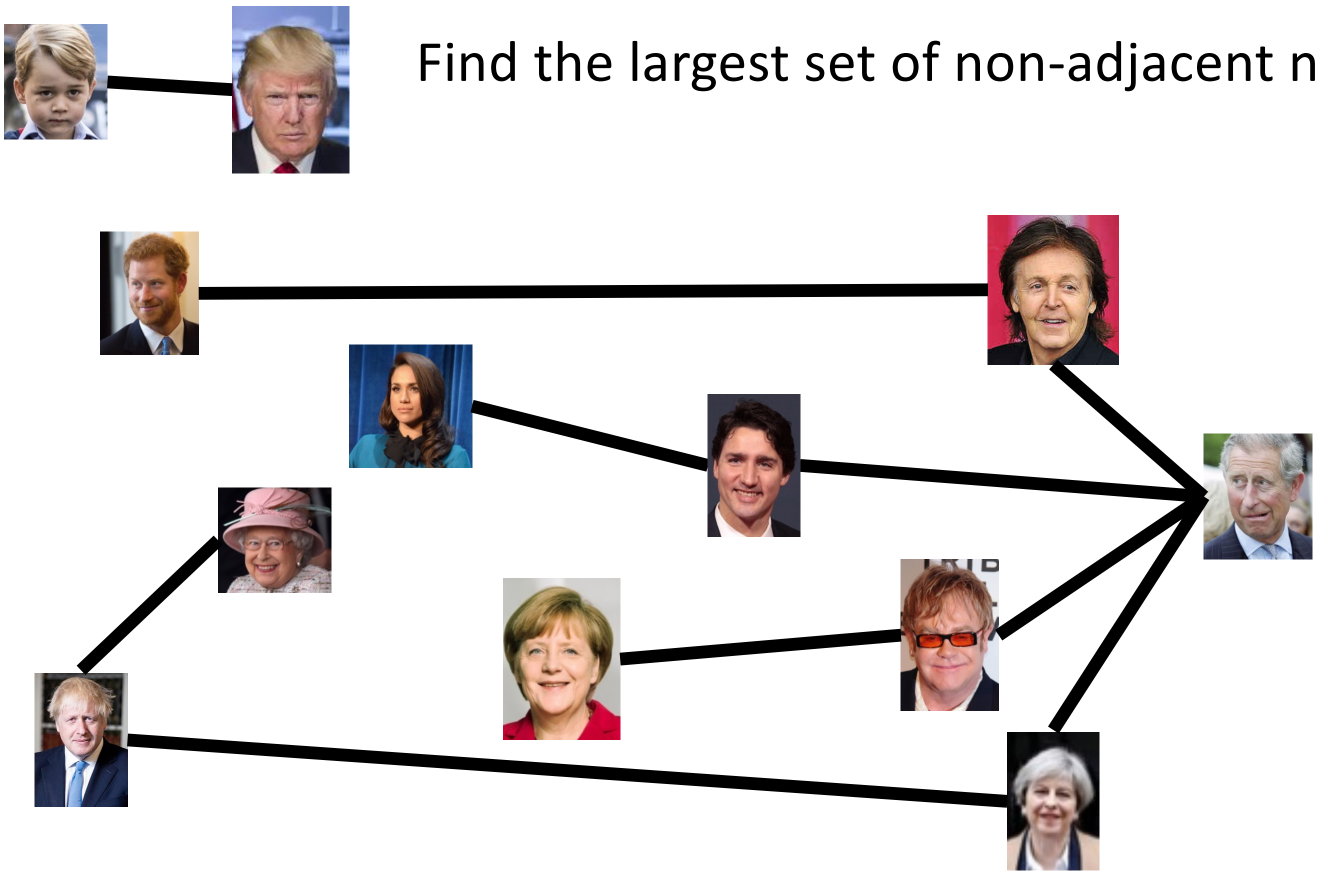
What is a Decision Problem?

Your response is maybe:
Groan! Do we really need to know?
Why do we need to care?

Turns out that the math and theory on NP-complete problems starts with decision problems.

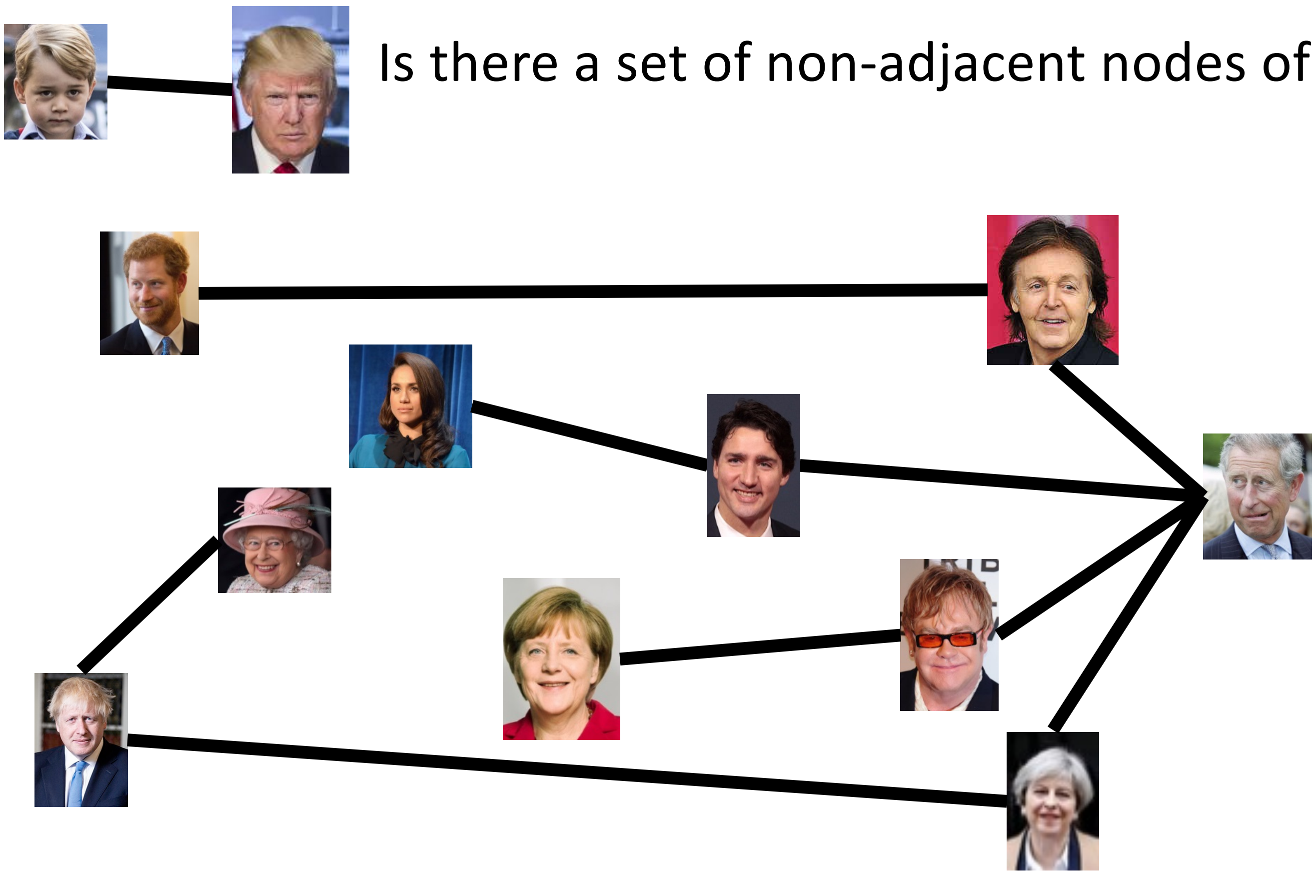
Max Independent Set

Find the largest set of non-adjacent nodes



k Independent Set

Is there a set of non-adjacent nodes of size k?



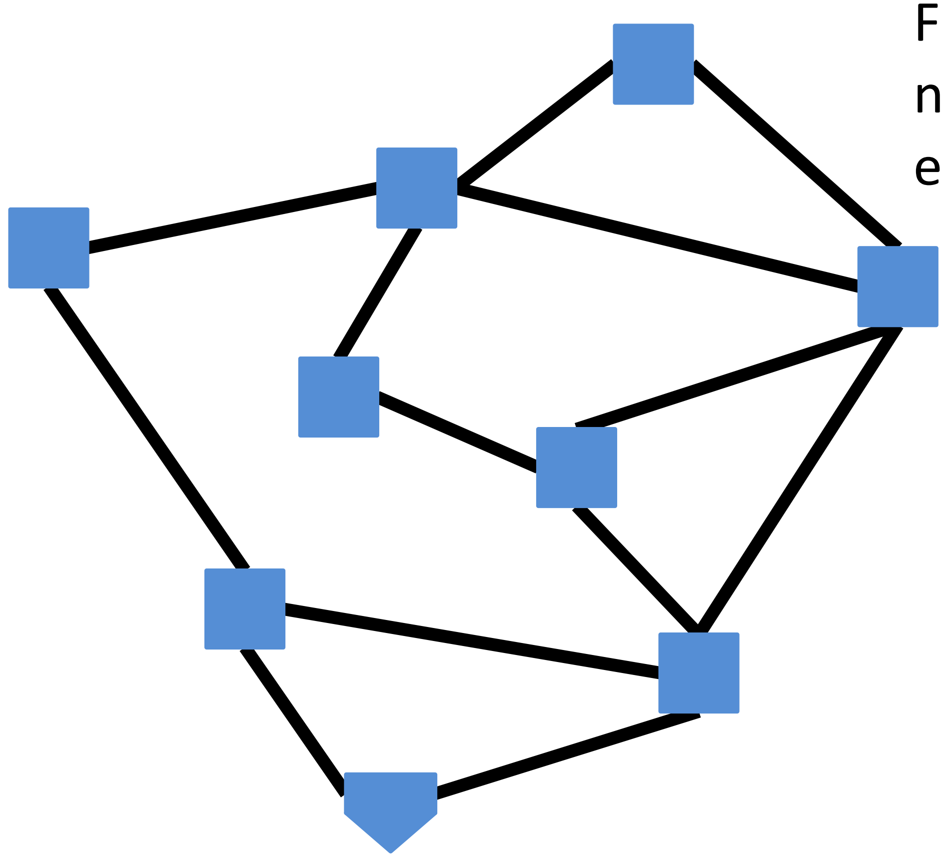
Maximum Independent Set

- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

k Independent Set

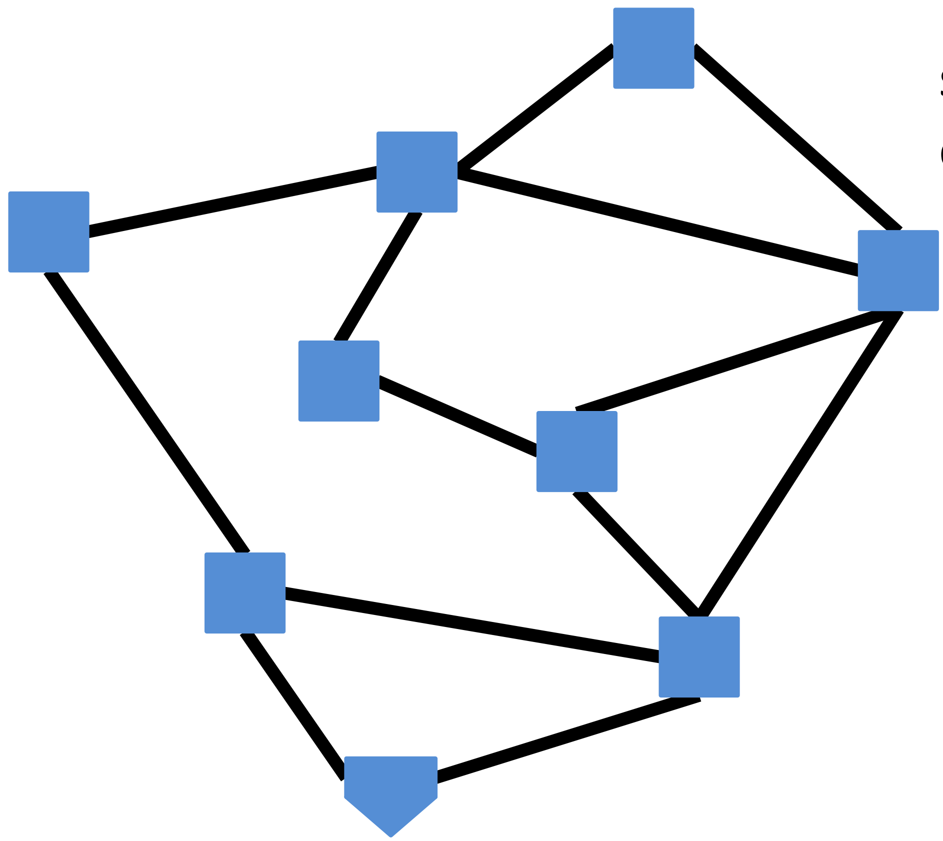
- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- k Independent Set Problem: Given a graph $G = (V, E)$ and a number k, **determine whether there is an independent set $\boldsymbol{S}$ of size $\boldsymbol{k}$**

Min Vertex Cover



Find the smallest set of nodes which covers every edge

k Vertex Cover



Is there a set of nodes of size k which covers every edge?

Minimum Vertex Cover

- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover: Given a graph $G = (V, E)$ find the minimum vertex cover C

k Vertex Cover

- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- k Vertex Cover: Given a graph $G = (V, E)$ and a number k, **determine whether there is a vertex cover $\boldsymbol{C}$ of size $\boldsymbol{k}$**

Problem Types

- Decision Problems:
  - Is there a solution?
    - Output is True/False
  - Is there a vertex cover of size k?
- Optimal Value Problems
  - E.g. What's the min k for k-vertex cover problem?
- Search Problems:
  - Find a solution
    - Output is complex
  - Give a vertex cover of size k
- Verification Problems:
  - Given a potential solution, is it valid?
    - Output is True/False
  - Is **this** a vertex cover of size k**?**

If we can solve this

Then we can solve this

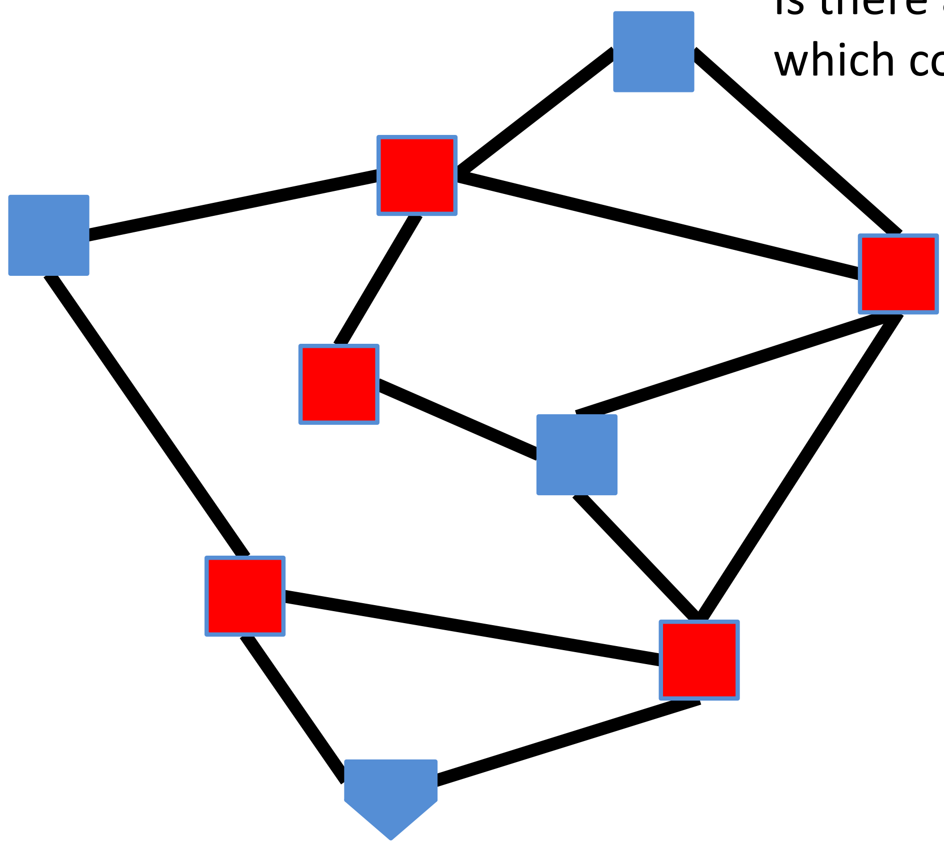
and this

Using a k-VertexCover decider to build a searcher

- Set $i = k - 1$
- Remove nodes (and incident edges) one at a time
- Check if there is a vertex cover of size i
  - If so, then that removed node was part of the k vertex cover, set $i = i - 1$
  - Else, it wasn't

Did I need this node to cover its edges to have a vertex cover of size k?

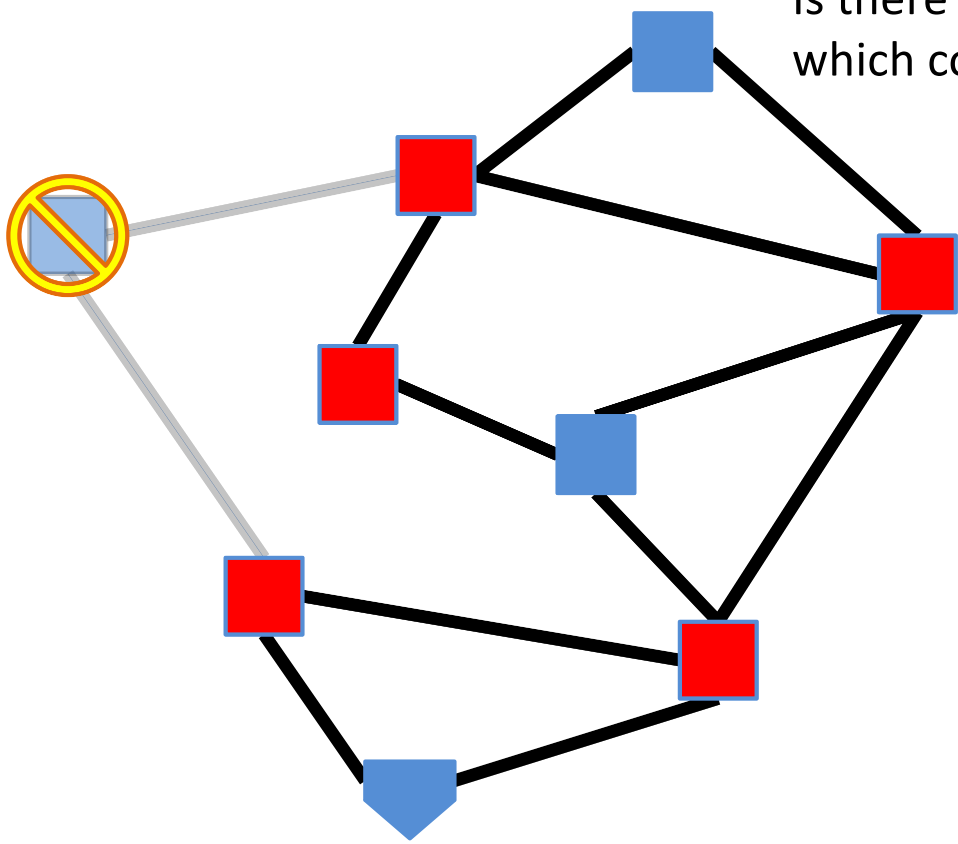
5 Vertex Cover (Decision)



Is there a set of nodes of size 5 which covers every edge?

Yes!

4 Vertex Cover (Decision)



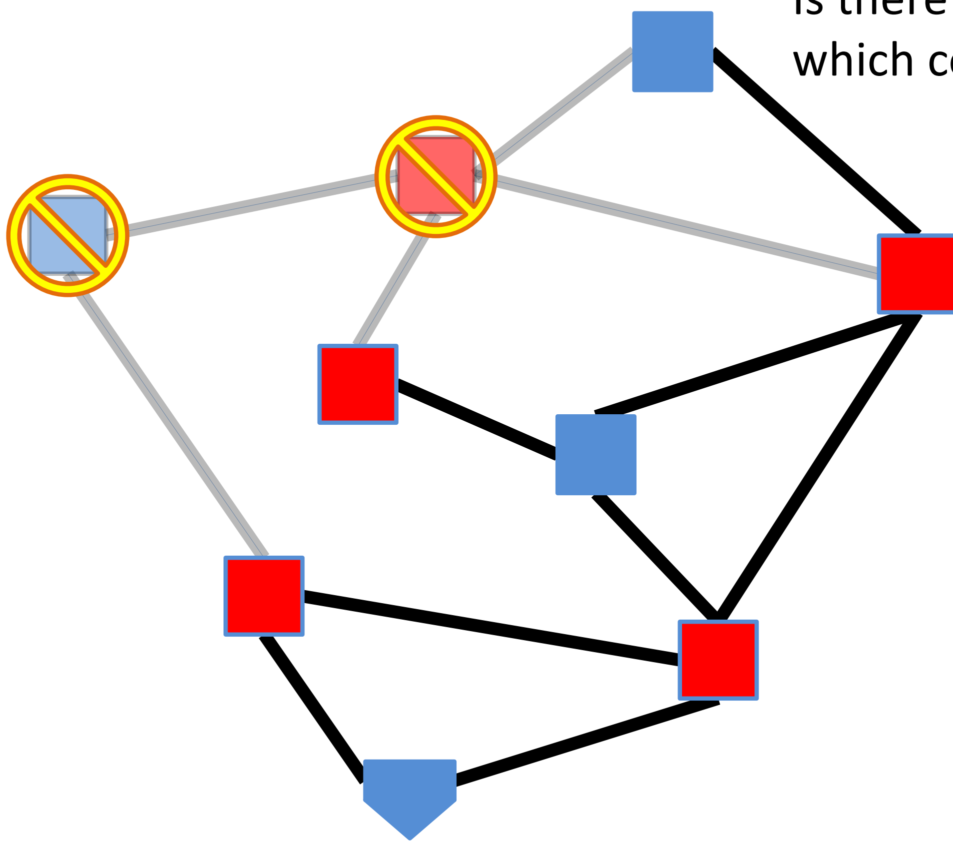
Is there a set of nodes of size 4 which covers every edge?

No!

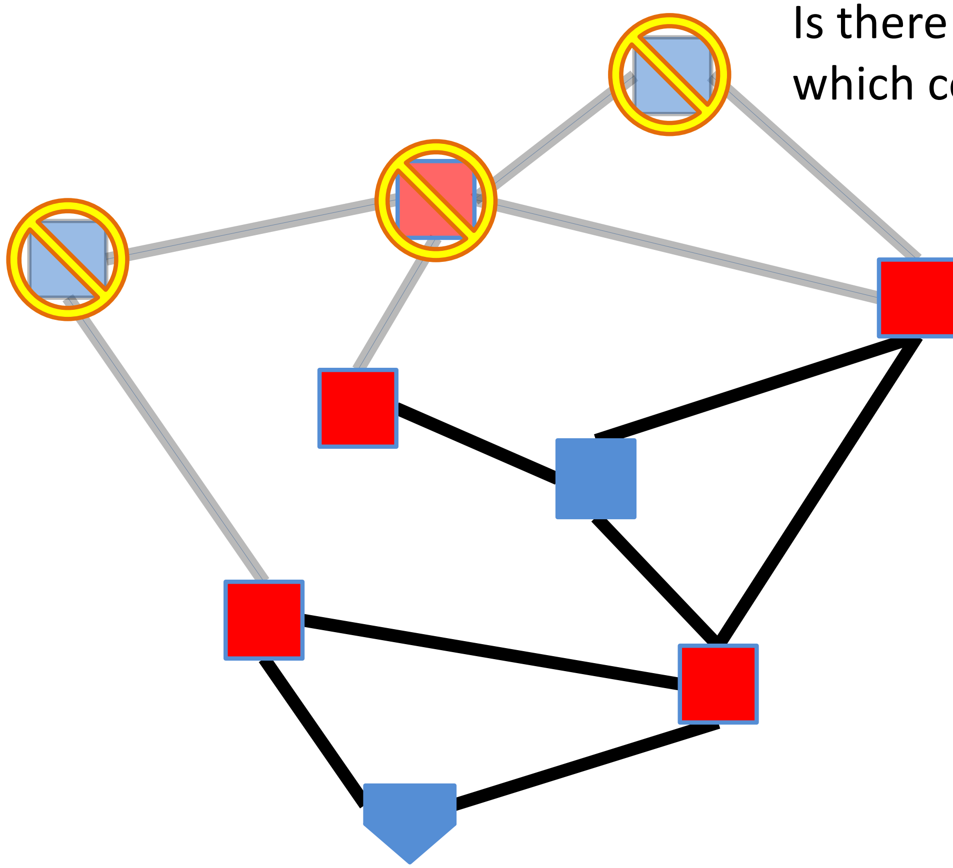
4 Vertex Cover (Decision)

Is there a set of nodes of size 4 which covers every edge?

Yes!



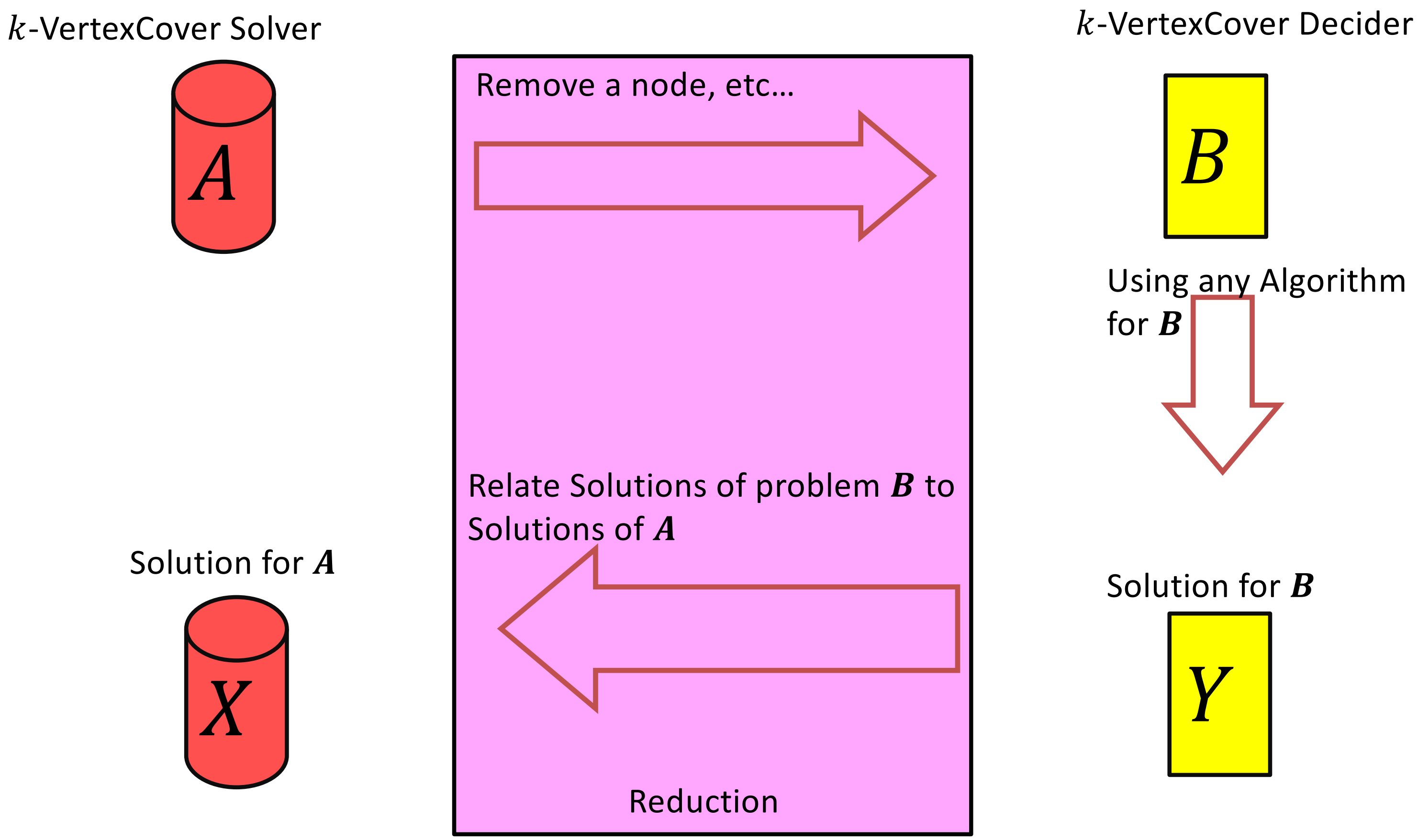
3 Vertex Cover (Decision)



Is there a set of nodes of size 3 which covers every edge?

No!

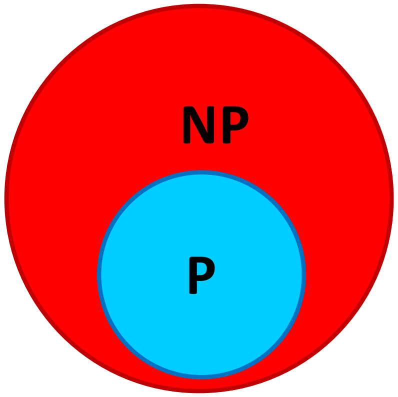
Reduction



k-VertexCover Solver
A
Solution for $\boldsymbol{A}$
X
Remove a node, etc…
Relate Solutions of problem $\boldsymbol{B}$ to Solutions of $\boldsymbol{A}$
Reduction
k-VertexCover Decider
B
Using any Algorithm for $\boldsymbol{B}$
Solution for $\boldsymbol{B}$
Y

P vs NP

- P
  - Deterministic Polynomial Time
  - Problems solvable in polynomial time
    - $O(n^p)$ for some number p
- NP
  - Non-Deterministic Polynomial Time
  - Problems verifiable in polynomial time
    - $O(n^p)$ for some number p
- Open Problem: Does P=NP?
  - Certainly $P \subseteq NP$



k-Independent Set is NP

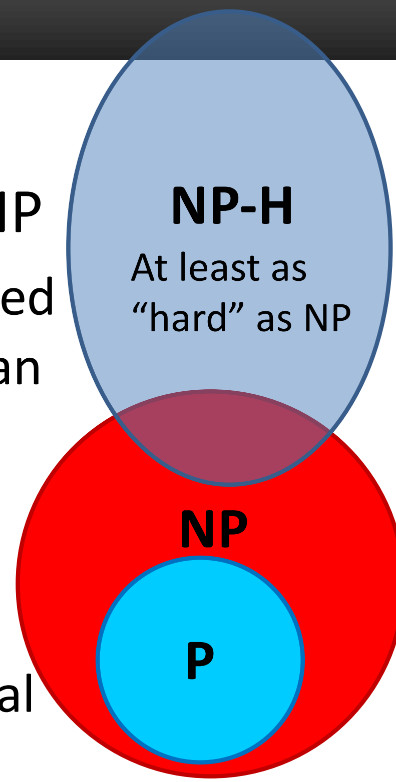
- To show: Given a potential solution, can we **verify** it in $O(n^p)$? $[n = V + E]$

How can we verify it?

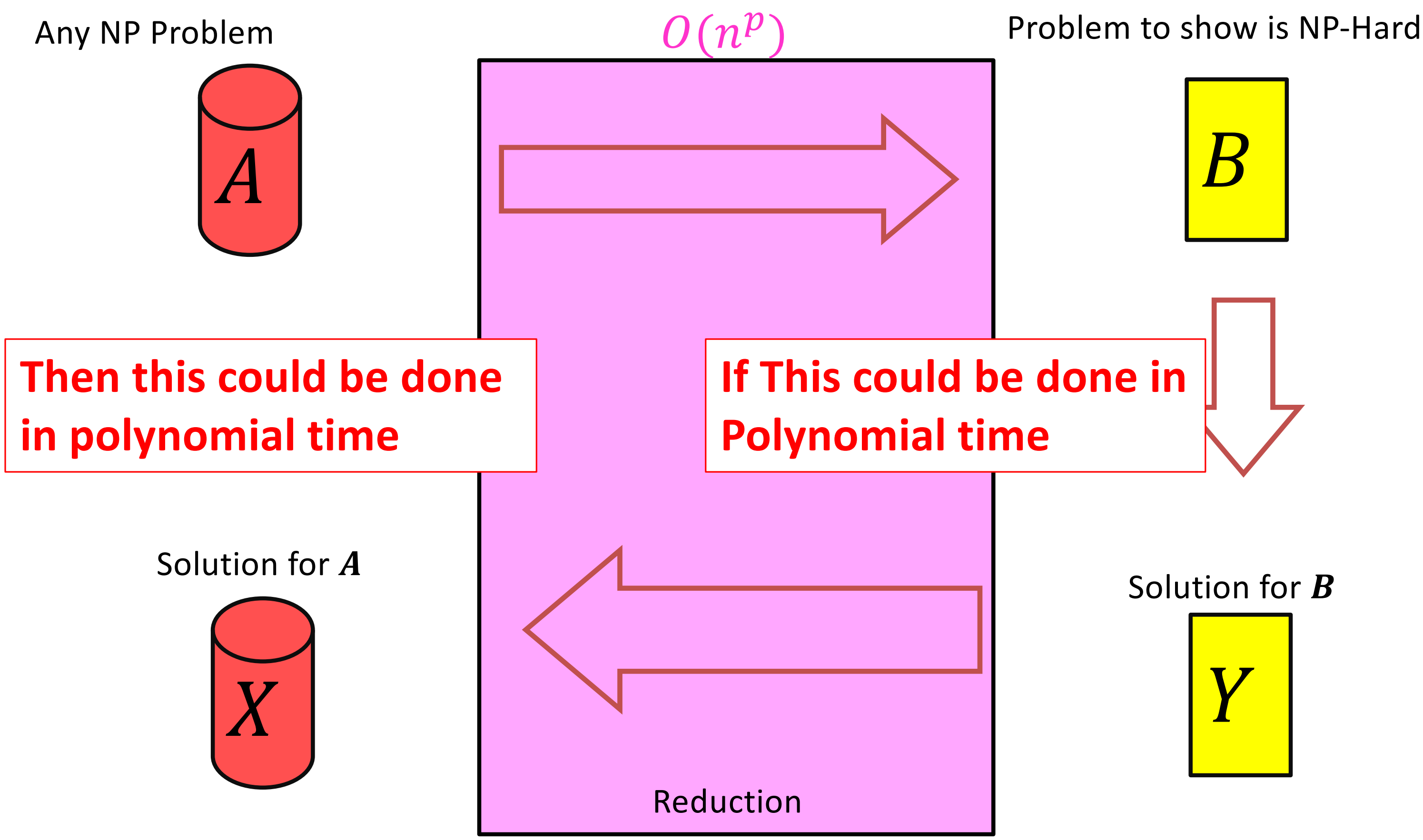
1. Check that it's of size k? Takes $O(V)$
2. Check that it's an independent set? Takes $O(V^2)$

NP-Hard

- How can we try to figure out if P=NP?
- Identify problems at least as "hard" as NP
  - If any of these "hard" problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard if $\forall A \in NP, A \leq_p B$
  - $A \leq_p B$ means A reduces to B in polynomial time



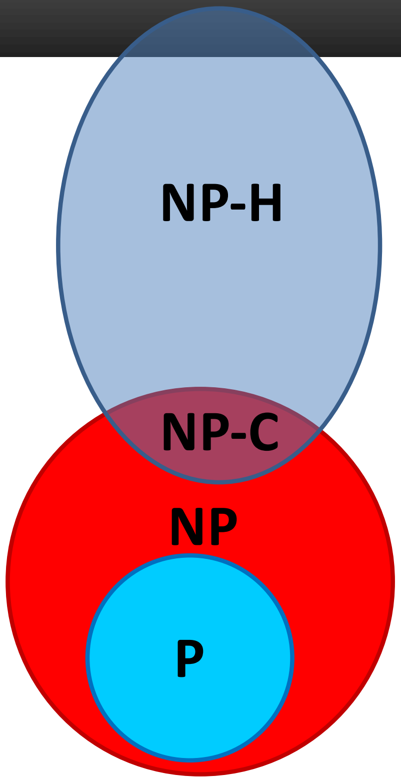
NP-Hardness Reduction



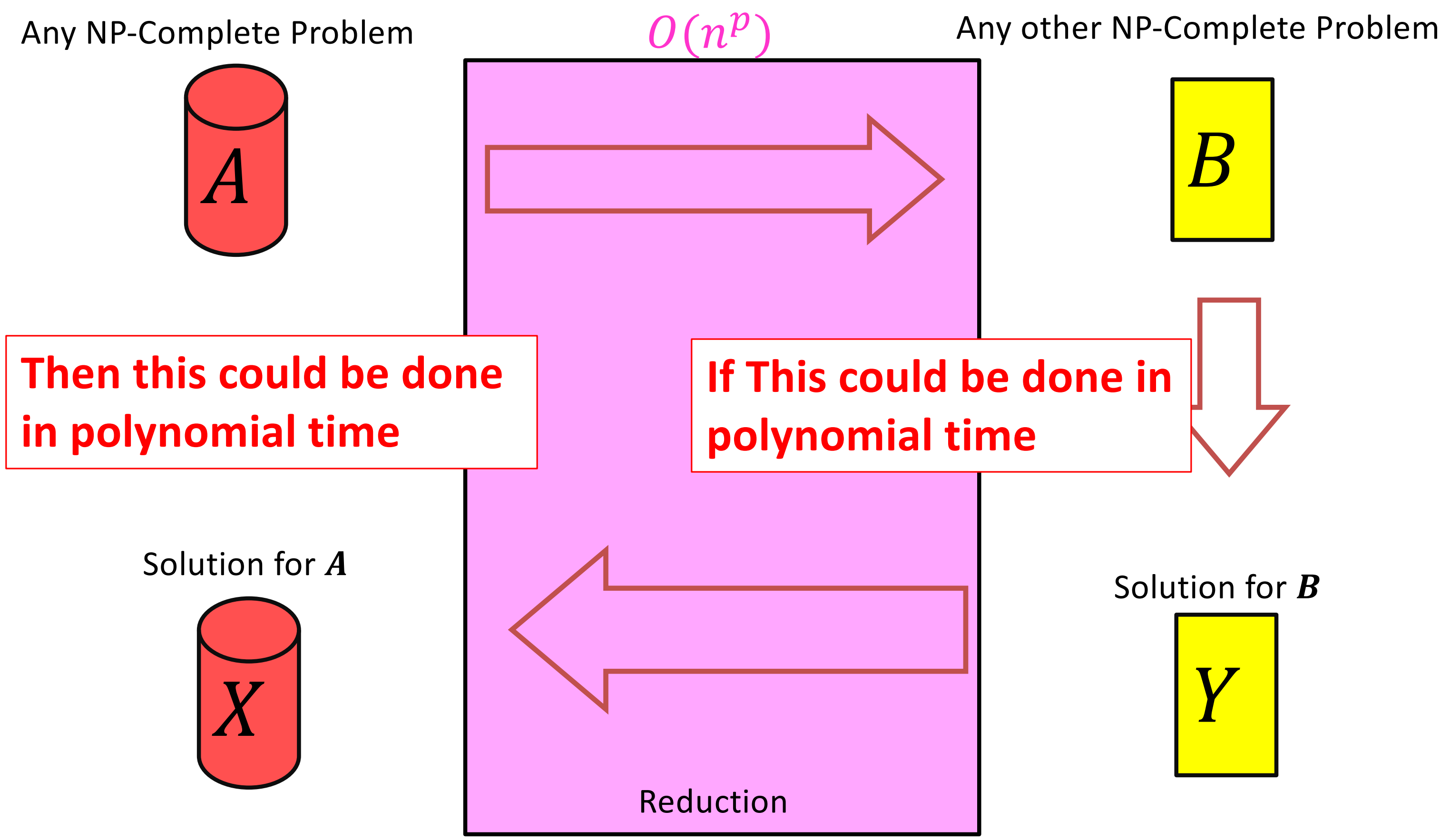
NP-Complete

- "Together they stand, together they fall"
- Problems solvable in polynomial time iff ALL NP problems are
- NP-Complete = NP ∩ NP-Hard
- **How to show a problem is NP-Complete?**
  - Show it belongs to NP
    - Give a polynomial time verifier
  - Show it is NP-Hard
    - Give a reduction from another NP-H problem

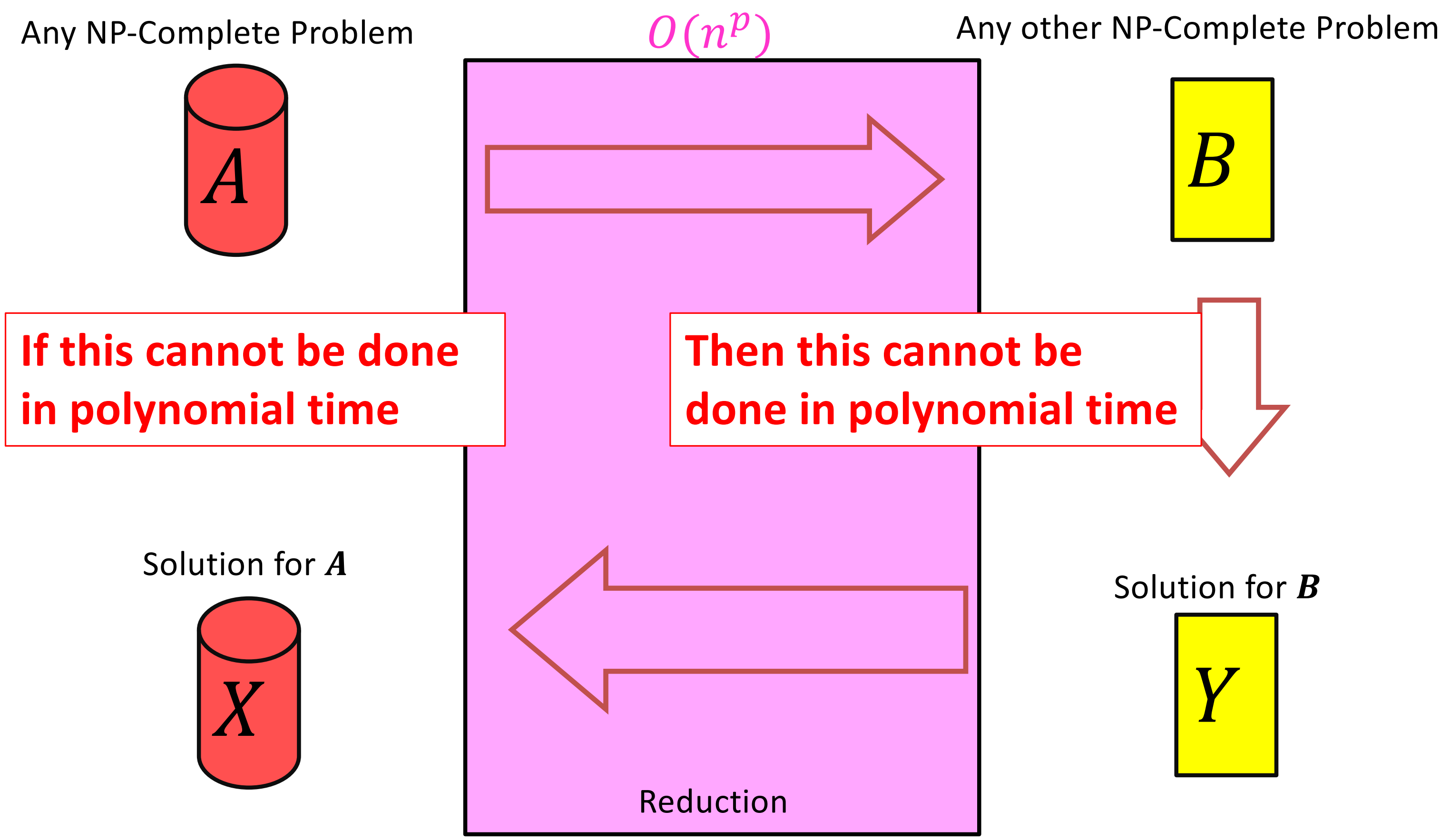
We now just need a FIRST NP-Hard problem



NP-Completeness



NP-Completeness



Wrap Up

- Reductions used to show “hardness” relationships between problems
- Intractable problems often reduce to each other
- Starting to define “classes” of problems based on complexity issues
  - P are problems that can be solved in polynomial time
  - NP are problems where a solution can be verified in polynomial time
  - NP-hard are problems that are at least as hard as anything in NP
  - NP-complete are NP-hard problems that “stand or fall together”